FIELD OF INVENTION

This invention relates to the provision of data over the Internet and, more particularly, to a system for ascertaining the demography of the users of the Internet.

BACKGROUND OF THE INVENTION

With the increased usage of the Internet comes the problem of deciding how and where to direct the information from the provider's point of view. With the advent of so-called "push" systems in which providers have direct access to the PC at the client side in a client/server application, it is increasingly important that the provider be provided with information so as to be able to direct the services to those users who are most likely to be interested.

In the past, the only type of demographic information that was available was to "guess" the usage of the system through use of sampling research data. In systems in which sampling research data is involved, a given research company will ask a major provider how many pieces of software were sold to the various users. Based on the data of the sales of the enabling software, the research company, utilizing mathematical techniques, provides demographic information to the provider based on a series of assumptions about the user.

However, the utilization of statistics alone based on the sales of software, for example, is not at all accurate in terms of providing the provider with targeted information as to the "real" demography of the user or client. For instance, it is impossible through statistics alone based on a single input such as sales to derive information relating to the CPU size and

speed at the user, hard disk space availalle, information relating to the network connection such as dial-up cable modem connection information and ISDN connections, a list of the inventories indicating the applications running on the particular computer involved, as well as peripherals such as sound cards connected to the computer at the client side. Moreover, there is no way to ascertain the log-in history for each of the Internet users, such that critical information for the providers is not existent.

Critical information which is not available from traditional research is infrastructure information for a particular PC, such as CPU power, viewer, sound card and Internet connection information.

What is meant by the term "viewer" is what type of protocol is being run on the machine such as MPEG, QUICKTIME, AVI, and PDF.

The sum total of this infrastructure information would be useful for the decision maker at the provider as to decide whether or not, for instance, it is worth the money to make 3 megabytes of MPEG video available for advertising based on the above infrastructure demography. Thus, it is impossible for this decision maker to ascertain whether the Internet advertisement delivery will be efficient and worth enough to justify the cost, much less, for instance, providing an Internet video advertisement, the cost of which must be justified by assuring a number of targeted viewers for the subject matter of the video advertisement.

SUMMARY OF THE INVENTION

In order to provide such needed data to a content provider that wants to use the Internet connection, in one embodiment, client software is installed at each PC which can detect the

infrastructure of the PC. The software which is provided by the provider to the user enables sensing CPU power, hard disk space, the applications running or installed, network connectivity and the log-in history. Since each client software has a unique serial number, sensing the serial number at the server side provides for rapid transfer and loading of a database with infrastructure data which is reported to the server periodically, for instance, every two seconds. At the server side, the database can be updated frequently to provide instant demographics of the particular user. It will be appreciated that the database can be filtered by such factors as location so that the provider can be apprised of what locations would be most interested in the content that the provider wishes to transmit. This permits the provider to be able to limit the broadcast of the data to selected locations.

It will be noted that the client software is delivered by the provider to the end user. By so doing, each provider is apprised of its own members, thus to provide the provider with the demographics of its own members or subscribers. This enables the content providers to be able to decide the content size to be delivered, as well as the viewer software, and makes the decision as to how much and what should be provided to an individual user tailored to the particular user's requirements.

For instance, in one operative embodiment, if a provider wants to make a two minute commercial, this can take as much eight megabytes to transmit. At this point, the provider must select what type of viewer software is required to play the eight megabytes of information, whether it is MPEG, QUICKTIME or some other format. By this manner, the provider can ascertain in real time whether or not the hard disk space is available at the user.

The distribution curve that is generatable through the utilization of the subject system, permits a bell curve to be

formed in which hard disk space can be presented in terms of the number of users. Assuming that 8 megabytes is required, it can be determined what percentage of the channels are occupied by the information to be transmitted, and thus the number of PC's that are available to receive the intended transmission. If, for instance, 8 megabytes represent 80% of the channels, then the provider may well be advised that there is a sufficient number of PCs that can receive the information to commit the resources to providing the content and transmitting it.

As will be appreciated, the subject system permits the content provider to make a decision as to whether or not to invest in a given project based on real-time demographics of users connected to the Internet. Note that in general, demographics are from members of the provider since the provider provides the client software to the end user. As a result, not only can content be tailored to the audience which could receive it, but critical decisions can be made as to whether to provide the content at all based on real-time sensing of the demography of the users.

The result is that by use of the subject system, providers can make business decisions such as the size of the video message, the viewer of the video/audio message, and the timing for hyper-advertisements through pulling or pushing at appropriate times. The subject system also makes possible other decisions which are critical to the economic utilization of the Internet. Furthermore, the demographic information permits marketing decisions as to where to sell the software and hardware based on the infrastructure data and the destination of the PC, namely its IP address.

In summary, in an Internet-based client/server application, a system is provided which detects demographics of a client including CPU power, hard disk space, applications installed, network connectivity and log-in history so as to provide this

infrastructure related information detailing client usage of the Internet to the service provider. In one embodiment, each user is provided with software having a unique serial number. Having the serial number, infrastructure data is checked at the client side and reported to the server periodically, with the server updating a database with the infrastructure data from each PC. In one embodiment, the database is filtered by factors such as location of the client and an indication of which providers delivered software to a client. After filtering, the service provider can obtain various demographics such as the demography of hard disk space, CPU power and viewers. In one embodiment, the demographics are used at the server to automatically select the contents to be transmitted to the particular client. Thus the provider can send the most appropriate contents to the most appropriate client based on demographic information of the client's infrastructure.

Moreover, having derived the above demographic information, in one embodiment, an automatic selection system uses the demographics to provide specially tailored contents to the client. Files can thus be tailored to the client's ability to receive the file or even as to what content should be delivered. Thus whether full frame video should be sent, whether audio should be sent, or indeed what format is appropriate can be selected.

BRIEF DESCRIPTION OF THE DRAWINGS

These and other features of the subject invention will be better understood taken in conjunction with the Detailed Description in conjunction with the figures of which:

Figure 1 is a block diagram of the subject system illustrating the ability to provide a database with real-time demographic information from the users, along with a filter

system to be able to tailor the demographic output to be the most useful to the provider;

Figure 2 is a graph illustrating the results of the subject system in which hard disk space is graphed against the number of users;

Figure 3 is a bar graph illustrating CPU power as a function of the number of users;

Figure 4 is a block diagram and flow chart illustrating one embodiment of the subject system indicating client-initiated data reporting based on client-installed software from the provider; and,

Figure 5 is a block diagram of the use of the demographic information for automatic delivery selection by the server.

DETAILED DESCRIPTION

Referring now to Figure 1, in the subject system, a provider 10 provides software to each of clients 14 which causes the associated computer connected to the Internet to transmit not only a serial number, but also the IP address, CPU information, hard disk space, network connection, a list of inventories, peripherals such as sound cards, and the log-in history associated with each individual client. Client and server are connected via the Internet, here diagrammatically illustrated at 16.

The information is passed to a database 18 at the provider which, optionally, is provided with a database filter 20 so that the data may be filtered as to location as illustrated at 22 or is further filtered by the provider as illustrated at 24. It will be appreciated that there are other filter functions that can be applied to the database for the filtering of the statistics garnered by the subject system.

If, as illustrated, the statistics come from members of the provider because the provider provides the client software, then the provider is provided with information relating to his members and his members alone.

The data, when retrieved, may be in the form of a graph 26 of hard disk space versus number of CPU's or, alternatively, can be provided in terms of CPU power as illustrated at 28, with the information being developed at the server side, here illustrated at 30, and with the information from the server side being delivered to the provider after it has been derived.

Referring now to Figure 2, a graph is shown of hard disk space versus the number of users which, as will be appreciated, is developed in a bell shaped curve 40 with one end of the curve illustrating the number of users having only 1 kilobyte of disk space and with the other end reflecting 1 gigabyte of disk space. As mentioned hereinbefore, it can be seen that with 8 megabytes of information to be transmitted as illustrated by line 42, shaded area 46 represents approximately 80% of the channels and thus, in one instance, over a million PC's having the capability of receiving 8 megabyte transmissions. From this graph, a decision maker can decide that it is worth the time, effort and money to invest in the aforementioned two minute advertising sequence, with the graph giving the provider an instant view of his audience.

Referring now to Figure 3, what is shown is a bar graph 50 which charts CPU power versus number of CPU's, knowing computer power can lead to deductions about peripherals. For instance, one can deduce whether or not there is a sound card associated with a given PC. Thus in terms of CPU power, one can deduce if there is a 155 megahertz 486 processor installed, and/or if a sound card is in existence, because such computers usually come with a sound card installed. This being the case, it can be assumed that multimedia transmissions can be handled by such a CPU.

Additionally, not only will the providers be provided with information regarding the capability of the particular CPU to receive multimedia transmissions, these user's are also a very good target for the sales of sound cards. As a result, messages advertising sound cards can be sent directly only to those users which have sufficient CPU power.

In general, the client software is delivered along with applications software by the provider and is transparent to the user. At the provider's election, data relating user's identity can be inhibited so that the system is a pure anonymous demographics system. The system can be made anonymous simply by sensing only the IP address of the user as opposed to the user's identity. Thus, while the user's identity remains anonymous, his buying habits and usage create a powerful tool to direct advertising and other content to the user based on his prior usage, the ability of a CPU to receive the intended message and other factors. Note, however, that providers may seek to provide targeted advertisement and content to a particular user, assuming that the provider has the user's permission to do so.

Referring now to Figure 4, a flow chart is presented in which server 2, here illustrated at 60, sends out to client 62 a client software package 64 which is installed as illustrated at 66 at the client's PC. The software checks the infrastructure data of the client as illustrated at 68 and reports the data as illustrated at 70 through the Internet through server 1, here illustrated at 72, which updates the data to its database as illustrated at 74. The output of the database is filtered at 76 based on functions such as provider ID or other factors. The filtered data is used at 78 to create a demography for the particular filtered factor such as hard disk space.

As illustrated at 80, the result of the demography is provided to provider by sending this demography to server 60 such that a decision maker 82 relying on data from Server 2 can

make the appropriate decisions as to what to send out, e.g. 4 MB MPEG, 300 kB PDF etc.

Referring now to Figure 5, having derived the infrastructure demographies of a client's PC, it is possible to tailor the contents and delivery from a server to a given client in a so-called "push" system. First, based on log-in history, hard disk space and a variety of factors such as available as demography 100, a content provider can decide whether the proposed content would be suitable to the infrastructure of the client. Based on infrastructure information relating to actual use of a client's PC, including applications previously run, it is possible to ascertain the user's willingness or receptiveness to receiving the proposed content. Thus the content need only be pushed to users who in all likelihood would welcome receipt.

Additionally, it is possible for the content provider to have several different versions of the content. One version might require MPEG compatibility and a sound card. Another version might be a reduced file size or just a document.

As can be seen from Figure 5, an automatic delivery system 104 is provided which selects the destination of the contents based on the ability of a client to receive a given version of the contents. This is done by matching the requirements of the particular version of the contents with the demography of the client as shown at 106. Assuming only one version 116 of the contents, version requirements 117 for this version are matched at module 106 with the infrastructure of the clients. For those clients having infrastructure which can handle the particular version, destination selection module 110 switches version 116 to the appropriate clients.

Thus, upon a match, selection module 110 selects the destinations most appropriate for particular version of use the contents, at which time the version 116 is transmitted directly to the client's PC in a push operation as illustrated at 112.

The result is the receipt of a version appropriate for the infrastructure of the particular client as illustrated at 114.

As mentioned above, for contents available in different versions, it is possible to select from different versions of the contents, here shown at 118 and 120, in order to match the pushed version to the client's infrastructure.

Each version has a set of corresponding version requirements 119 and 121. These version requirements are supplied to matching module 106 which determines not which clients can receive a given version, but rather which versions can be sent to which clients. Selection module 110 then couples the appropriate version to the appropriate clients in a push operation.

What is now presented is a series of programs and C++ with the first of the programs being that which is provided to the client to query his particular PC and with the second program illustrating the retrieval of the data, the creation of the demography, and the transmission of the demography over the net to the server associated with the content provider:

Appendix

NOT PART of Specification

DO NOT PRINT

DO NOT PRINT

```
// SystemInventory.cpp : Miscellaneous utility routines.

#include "stdafx.h"
#include "stdlib.h"
#include <assert.h> // JRW
#include <fstream.h>

//#include <shlguid.h>
extern "C" {
#include <winnetwk.h>
#include <shellapi.h>
#include <shlobj.h>
#include <objbase.h>
#include <initguid.h>
#include <mmsystem.h>    // JRW
}

#include "winnls.h"

#include "atpattrib.h"  // For GetSystemInventory ...
#include "Utility.h"


#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// Compute total free filespace on a drive ...

DWORDLONG GetFreeFilespace(const char * driveRootPath)
{
    DWORDLONG freeSpace;
    DWORD   sectorsPerCluster;
    DWORD   bytesPerSector;
    DWORD   numberFreeClusters;
    DWORD   numberTotalClusters;
    if (!GetDiskFreeSpace(driveRootPath,
            &sectorsPerCluster,  &bytesPerSector,
            &numberFreeClusters, &numberTotalClusters))
        freeSpace = DWORDLONG(-1);
    else{
        freeSpace = sectorsPerCluster;
        freeSpace *= bytesPerSector;
        freeSpace *= numberFreeClusters;
        }
    return freeSpace;
}
// Get a big inventory of this system:
// ... this is a bit of a grab-bag, but it is convenient for
// getting all the information at once to set up to the server ...

#define FANCY_INVENTORY TRUE

void GetSystemInventory(AtpAttrib & inventory)
{
    CString tmp;
```

```
// Amount of physical RAM ...

    {
    MEMORYSTATUS memoryStatus;
    memset(&memoryStatus,0,sizeof(memoryStatus));
    memoryStatus.dwLength = sizeof(memoryStatus);
    GlobalMemoryStatus(&memoryStatus);
    tmp.Format("PhysicalRAM=\"%ld\"",memoryStatus.dwTotalPhys);
    inventory.ParseLine(tmp);
    }

// Amount of free disk temp space ...

    {
    char * pathBuffer = NULL;
    DWORD bufSize = GetTempPath(0, pathBuffer) + 5;
    pathBuffer = new char[bufSize];
    if (0 == GetTempPath(bufSize, pathBuffer))
        {
HACKassert("failure on GetTempPath", FALSE);
        inventory.ParseLine("TempDirectoryFilespace=\"-1\"");
        }
    else{
        CString driveLetter(pathBuffer);
        driveLetter = driveLetter.Left(3);
        tmp.Format("TempDirectoryFilespace=\"%Ld\"",
                                GetFreeFilespace(driveLetter));
        inventory.ParseLine(tmp);
        }
    delete [] pathBuffer;
    }

// Amount of free disk space in document directory ...

    {
// Note: assumes drive letter in start of string:
// is there some system call we can use to get it from a pathname
// without an explicit drive letter?
    CString pathName(inventory["IDF_DIRECTORY"]);
    pathName = pathName.Left(3);
    tmp.Format("IDF_DIRECTORYFilespace=\"%Ld\"", GetFreeFilespace(pathName));
    inventory.ParseLine(tmp);
    }

// OS version number:

    OSVERSIONINFO osVersionInfo;     // We will use this later ...
    {
    memset(&osVersionInfo,0,sizeof(osVersionInfo));
    osVersionInfo.dwOSVersionInfoSize = sizeof(osVersionInfo);
    if (0 != GetVersionEx(&osVersionInfo))
        {
        char * osName = NULL;
        if      (osVersionInfo.dwPlatformId == VER_PLATFORM_WIN32_NT)
            osName  = "WinNT";  // WinNT Yes, Win95 No, Win32s No
        else if (osVersionInfo.dwPlatformId == VER_PLATFORM_WIN32_WINDOWS)
            osName  = "Win95";  // WinNT Yes, Win95 Yes, Win32s No
        else if (osVersionInfo.dwPlatformId == VER_PLATFORM_WIN32s)
            osName  = "Win32s ";  // WinNT Yes, Win95 Yes, Win32s No
        else
```

```
            osName  = "Windows(Unknown)";
        tmp.Format("%s %d.%d: Build %d: %s",
                osName,
                osVersionInfo.dwMajorVersion,
                osVersionInfo.dwMinorVersion,
                osVersionInfo.dwBuildNumber,
                osVersionInfo.szCSDVersion);
        tmp = "OSVersion=" + AtpAttrib::QuoteString(tmp);
        inventory.ParseLine(tmp);
        }
    }

// Is there a CD-ROM? (Return the drive letter, else empty)
// Note: if more than one, we only count the first one.

    {
    char driveName[] = "Z:";
    tmp.Format("CD_ROM=\"%s\"", "");
    for (driveName[0] = 'a'; driveName[0] <= 'z'; driveName[0]++)
        {
        if (DRIVE_CDROM == GetDriveType(driveName))
            {
            tmp.Format("CD_ROM=\"%s\"", driveName);
            break;
            }
        }
    inventory.ParseLine(tmp);
    }

// Network present, Slow machine, etc. from GetSystemMetrics.

    {
    int value;
    value = GetSystemMetrics(SM_NETWORK);
    tmp.Format("NetworkPresent=\"%s\"", ((value & 0x01)!=0) ? "TRUE" : "FALSE");
    inventory.ParseLine(tmp);

    value = GetSystemMetrics(SM_SLOWMACHINE);
    tmp.Format("SlowMachine=\"%s\"", value ? "TRUE" : "FALSE");
    inventory.ParseLine(tmp);
    }

// Stuff from SystemParametersInfo: Windows Plus! extensions ...

    {
    BOOL bStat = SystemParametersInfo(SPI_GETWINDOWSEXTENSION, 1, NULL, 0);
    tmp.Format("WindowsPLUS=\"%s\"", bStat ? "TRUE" : "FALSE");
    inventory.ParseLine(tmp);
    }

// Stuff from GetSystemInfo: CPU type, etc.

    {
    CString tmp, tmp2;
    SYSTEM_INFO systemInfo;
    memset(&systemInfo,0,sizeof(systemInfo));
    GetSystemInfo(&systemInfo); // Doesn't work on Win32s.

// wProcessorArchitecture:
#ifdef  FANCY_INVENTORY
```

```
    if       (systemInfo.wProcessorArchitecture==PROCESSOR_ARCHITECTURE_INTEL)
        tmp = "PROCESSOR_ARCHITECTURE_INTEL";
    else if (systemInfo.wProcessorArchitecture==PROCESSOR_ARCHITECTURE_MIPS)
        tmp = "PROCESSOR_ARCHITECTURE_MIPS";
    else if (systemInfo.wProcessorArchitecture==PROCESSOR_ARCHITECTURE_ALPHA)
        tmp = "PROCESSOR_ARCHITECTURE_ALPHA";
    else if (systemInfo.wProcessorArchitecture==PROCESSOR_ARCHITECTURE_PPC)
        tmp = "PROCESSOR_ARCHITECTURE_PPC";
    else if (systemInfo.wProcessorArchitecture==PROCESSOR_ARCHITECTURE_UNKNOWN)
        tmp = "PROCESSOR_ARCHITECTURE_UNKNOWN";
    else
        tmp = "PROCESSOR_ARCHITECTURE_UNUNDEFINED";
    tmp2.Format("%s", (const char *) tmp);
    tmp = "ProcessorArchitecture=" + AtpAttrib::QuoteString(tmp2);
    inventory.ParseLine(tmp);
#endif
    tmp2.Format("%d",systemInfo.wProcessorArchitecture);
    tmp = "SYSTEM_INFO.wProcessorArchitecture=" + AtpAttrib::QuoteString(tmp2);
    inventory.ParseLine(tmp);


// dwNumberOfProcessors:
#ifdef  FANCY_INVENTORY
    tmp.Format("NumberOfProcessors=\"%ld\"", systemInfo.dwNumberOfProcessors);
    inventory.ParseLine(tmp);
#endif
    tmp.Format("SYSTEM_INFO.dwNumberOfProcessors=\"%ld\"",
               systemInfo.dwNumberOfProcessors);
    inventory.ParseLine(tmp);

// wProcessorType:  Supposedly only used in Win95 ...
#ifdef  FANCY_INVENTORY
    if       (systemInfo.dwProcessorType == PROCESSOR_INTEL_386)
        tmp = "PROCESSOR_INTEL_386";
    else if (systemInfo.dwProcessorType == PROCESSOR_INTEL_486)
        tmp = "PROCESSOR_INTEL_486";
    else if (systemInfo.dwProcessorType == PROCESSOR_INTEL_PENTIUM)
        tmp = "PROCESSOR_INTEL_PENTIUM";
    else if (systemInfo.dwProcessorType == PROCESSOR_MIPS_R4000)
        tmp = "PROCESSOR_MIPS_R4000";
    else if (systemInfo.dwProcessorType == PROCESSOR_ALPHA_21064)
        tmp = "PROCESSOR_ALPHA_21064";
    else
        tmp = "PROCESSOR_UNKNOWN";
    tmp2.Format("%s", (const char *) tmp);
    tmp = "ProcessorType=" + AtpAttrib::QuoteString(tmp2);
    inventory.ParseLine(tmp);
#endif
    tmp2.Format("%d", systemInfo.dwProcessorType);
    tmp = "SYSTEM_INFO.dwProcessorType=" + AtpAttrib::QuoteString(tmp2);
    inventory.ParseLine(tmp);

// wProcessorLevel: gets kind of fancy for different models ...
// Note: not used on Win95 ...
    if (osVersionInfo.dwPlatformId != VER_PLATFORM_WIN32_WINDOWS)
    {
#ifdef  FANCY_INVENTORY
    tmp = "";
    if       (systemInfo.wProcessorArchitecture == PROCESSOR_ARCHITECTURE_INTEL)
        {   // 3 == 386, 4 = 486, 5 = Pentium
```

```
        if      (systemInfo.wProcessorLevel == 3)
            tmp = "Intel 80386";
        else if (systemInfo.wProcessorLevel == 4)
            tmp = "Intel 80486";
        else if (systemInfo.wProcessorLevel == 5)
            tmp = "Intel Pentium";
        else
            tmp = "Intel Unknown";
        }
    else if (systemInfo.wProcessorArchitecture == PROCESSOR_ARCHITECTURE_MIPS)
        {   // 4 = R4000
        if (systemInfo.wProcessorLevel == 5)
            tmp = "MIPS R4000";
        else
            tmp = "MIPS Unknown";
        }
    else if (systemInfo.wProcessorArchitecture == PROCESSOR_ARCHITECTURE_ALPHA)
        {   // 21064, 21066, 21164
        if      (systemInfo.wProcessorLevel == 21064)
            tmp = "Alpha 21064";
        else if (systemInfo.wProcessorLevel == 21066)
            tmp = "Alpha 21066";
        else if (systemInfo.wProcessorLevel == 21166)
            tmp = "Alpha 21166";
        else
            tmp = "Alpha Unknown";
        }
    else if (systemInfo.wProcessorArchitecture == PROCESSOR_ARCHITECTURE_PPC)
        {   // 1 601, 3 603, 4 604, 6 603+, 9 604+, 20 620
        if      (systemInfo.wProcessorLevel == 1)
            tmp = "PPC 601";
        else if (systemInfo.wProcessorLevel == 3)
            tmp = "PPC 603";
        else if (systemInfo.wProcessorLevel == 4)
            tmp = "PPC 604";
        else if (systemInfo.wProcessorLevel == 6)
            tmp = "PPC 603+";
        else if (systemInfo.wProcessorLevel == 9)
            tmp = "PPC 604+";
        else if (systemInfo.wProcessorLevel == 20)
            tmp = "PPC 620";
        else
            tmp = "PPC Unknown";
        }
// Include raw value, just in case Micro$oft extends something on us ...
    tmp2.Format("%s", (const char *) tmp);
    tmp = "ProcessorLevel=" + AtpAttrib::QuoteString(tmp2);
    inventory.ParseLine(tmp);
#endif
    tmp2.Format("%ld",systemInfo.wProcessorLevel);
    tmp = "SYSTEM_INFO.wProcessorLevel=" + AtpAttrib::QuoteString(tmp2);
    inventory.ParseLine(tmp);
    }

// wProcessorRevision: gets really fancy for different models ...
// Note: not used on Win95 ...
    if (osVersionInfo.dwPlatformId != VER_PLATFORM_WIN32_WINDOWS)
    {
#ifdef  FANCY_INVENTORY
    tmp = "";
```

```
if      (systemInfo.wProcessorArchitecture == PROCESSOR_ARCHITECTURE_INTEL)
    {   // 3 == 386, 4 = 486, 5 = Pentium
    if ((systemInfo.wProcessorLevel == 3) ||
        (systemInfo.wProcessorLevel == 4))
        {
        if ((systemInfo.wProcessorRevision & 0xFF00) == 0xFF00)
            tmp.Format("Model %d Step %d",
                ((systemInfo.wProcessorRevision & 0x00F0) >> 4) - 0xA,
                 (systemInfo.wProcessorRevision & 0x000F) >> 4);
        else
            tmp.Format("Model %c Step %d",
                ((systemInfo.wProcessorRevision & 0xFF00) >> 8) + 'A',
                 (systemInfo.wProcessorRevision & 0x00FF));
        }
    else if (systemInfo.wProcessorLevel == 5)
        tmp.Format("Model %d Step %d",
                ((systemInfo.wProcessorRevision & 0xFF00) >> 8),
                 (systemInfo.wProcessorRevision & 0x00FF));
    else
        tmp.Format("UNKNOWN(%d)", systemInfo.wProcessorRevision & 0xFF00);
    }
 else if (systemInfo.wProcessorArchitecture == PROCESSOR_ARCHITECTURE_MIPS)
    {   // 4 = R4000
    tmp.Format("Revision %d", systemInfo.wProcessorRevision & 0xFF00);
    }
 else if (systemInfo.wProcessorArchitecture == PROCESSOR_ARCHITECTURE_ALPHA)
    {   // 21064, 21066, 21164
    tmp.Format("Model %c Pass %d",
                ((systemInfo.wProcessorRevision & 0xFF00) >> 8) + 'A',
                 (systemInfo.wProcessorRevision & 0x00FF));
    }
 else if (systemInfo.wProcessorArchitecture == PROCESSOR_ARCHITECTURE_PPC)
    // 1 601, 3 603, 4 604, 6 603+, 9 604+, 20 620
    {
    tmp.Format("Version %02d.%02d",
                ((systemInfo.wProcessorRevision & 0xFF00) >> 8),
                 (systemInfo.wProcessorRevision & 0x00FF));
    }
 tmp2.Format("%s", (const char *) tmp);
 tmp = "ProcessorRevision=" + AtpAttrib::QuoteString(tmp2);
 inventory.ParseLine(tmp);
#endif
 tmp2.Format("%d",systemInfo.wProcessorRevision);
 tmp = "SYSTEM_INFO.wProcessorRevision=" + AtpAttrib::QuoteString(tmp2);
 inventory.ParseLine(tmp);
 }
 }

// Computer name, current user name

 {
 DWORD dwSize;
 char nameBuf[MAX_COMPUTERNAME_LENGTH+4096];
 dwSize = sizeof(nameBuf) -1 ;
 if (GetComputerName(nameBuf,&dwSize))
     inventory.ParseLine("ComputerName="  + AtpAttrib::QuoteString(nameBuf));

 dwSize = sizeof(nameBuf) -1 ;
 if (GetUserName(nameBuf,&dwSize))
     inventory.ParseLine("LogonUserName=" + AtpAttrib::QuoteString(nameBuf));
```

```
    }

// Existence of sound-output devices:
// For now, we just want to know that there is an output device for sound,
// not the details of the device.  We just return the number of such
// output devices (zero if no sound output)
// We don't care about microphone input for now: we think of our business
// as one of distributing documents to be "viewed", not general software.

    {
    UINT devCount;
    devCount     = midiOutGetNumDevs();
    tmp.Format("midiOutGetNumDevs=\"%ld\"", DWORD(devCount));
    inventory.ParseLine(tmp);

    devCount     = waveOutGetNumDevs();
    tmp.Format("waveOutGetNumDevs=\"%ld\"", DWORD(devCount));
    inventory.ParseLine(tmp);

    devCount     = auxGetNumDevs();
    tmp.Format("auxGetNumDevs=\"%ld\"", DWORD(devCount));
    inventory.ParseLine(tmp);
    }

}

// Support routine for CheckDocumentOpenAssociation:
// Get default value for a key, if it is a string.
// Returns TRUE  if there is one,
// Returns FALSE if none, or no such key, or default is not a string.

BOOL GetDefaultStringValue(const char * keyName, CString & defaultValue)
{
    HKEY hKey;
    LONG status;
    status = RegOpenKeyEx(
        HKEY_CLASSES_ROOT,                          // open key
        keyName,                                    // subkey
        0,                                          // reserved
        KEY_READ,                                   // security
        &hKey);
    if (status != ERROR_SUCCESS)
        return FALSE;              // No such key.

// Find out how many values to scan ...

    DWORD dwValueCount;
    DWORD dwMaxNameLen;
    DWORD dwMaxValueLen;
    status = RegQueryInfoKey(
            hKey,
            NULL, NULL,             // Class name and len: not used here.
            NULL,                   // Reserved.
            NULL,                   // Number of subkeys
            NULL,                   // Longest subkey name length
            NULL,                   // Longest class  name length
            &dwValueCount,
            &dwMaxNameLen,
            &dwMaxValueLen,
            NULL,                   // Security descriptor
```

```
            NULL);                          // Last time written
    if (status != ERROR_SUCCESS)
        {
        SetLastError(status);
HACKassert("Error on RegQueryInfoKey",status == ERROR_SUCCESS);
        throw;     // XXX Should be a different exception.
        }

    char * nameBuffer    = new char [dwMaxNameLen +2];
    char * valueBuffer   = new char [dwMaxValueLen+2];
    DWORD dwIndex, dwType;
    BOOL bFoundMatch = FALSE;
    for (dwIndex = 0; dwIndex < dwValueCount; dwIndex++)
        {
        DWORD  nameBufferSize  = dwMaxNameLen +1;
        DWORD  valueBufferSize = dwMaxValueLen+1;
        status = RegEnumValue(
                    hKey,                   // Key to enumerate
                    dwIndex,                // index of value
                    nameBuffer,
                    &nameBufferSize,
                    NULL,                   // reserved.
                    &dwType,                // buffer type code.
                    (unsigned char *) valueBuffer, // value data buffer
                    &valueBufferSize);      // value data buffer size.
        if (status != ERROR_SUCCESS)        // Should always work: we got
            {                               // the count and size values already.
            SetLastError(status);
HACKassert("Error on RegEnumValue",status == ERROR_SUCCESS);
            throw;     // XXX Should be a different exception.
            }
        if (dwType != REG_SZ)
            continue;          // default value is always a string.

// See if this is the default value ...

        if (strcmp(nameBuffer,"") != 0)
            continue;          // No: default value has empty string for a name

        defaultValue = valueBuffer;
        bFoundMatch = TRUE;
        break;
        }

    delete [] nameBuffer;
    delete []valueBuffer;
    status = RegCloseKey(hKey);
    hKey = NULL;
    if (status != ERROR_SUCCESS)
        {
        SetLastError(status);
HACKassert("Cannot close hKey handle",status == ERROR_SUCCESS);
        throw;     // XXX Should be a different exception.
        }

    return bFoundMatch;
}

// Support routine for GetAllDocumentOpenAssociations:
// returns TRUE if these an "open" class association in the registry like this.
```

```
// "open" command line passed back in openAppEntry.

BOOL CheckDocumentOpenAssociation(char * extensionName, CString & openAppEntry)
{
    openAppEntry = "XXXAppName";     // Debugging hack.

    CString className;
    if (!GetDefaultStringValue(extensionName, className))
        return FALSE;        // No matching entry for that file extension.

// OK, look for the "open" command line entry under this key name ...

    className += "\\shell\\open\\command";
    if (!GetDefaultStringValue(className, openAppEntry))
        return FALSE;        // No matching entry for that file extension.

    return TRUE;
}

// Routine to get a list of all the document types that have
// a "viewer", i.e. an association for "open" in the class registry.

void GetAllDocumentOpenAssociations(AtpAttrib & associationList)
{
// Strategy:
// Search for keys starting with a "."
// For each such key ...
//      Look for the <No Name> REG_SZ application name
//      Find the registry entry named that "application name"
//      Check for a sub-key
//          "application name"\shell\open\command
//          ... if one exists, then note the entry in our association list.
// Note:
// Keys look like they are lexigraphically ordered, but on Windows95,
// they aren't: it's a fake done by the "RegEdit" program.
// Find out how big a name buffer to allocate ...

    LONG status;
    DWORD dwMaxNameLen;
#ifdef DID_NOT_WORK
    DWORD dwValueCount;
    DWORD dwMaxValueLen;
    status = RegQueryInfoKey(
            HKEY_CLASSES_ROOT,
            NULL, NULL,                 // Class name and len: not used here.
            NULL,                       // Reserved.
            NULL,                       // Number of subkeys
            NULL,                       // Longest subkey name length
            NULL,                       // Longest class  name length
            &dwValueCount,
            &dwMaxNameLen,
            &dwMaxValueLen,
            NULL,                       // Security descriptor
            NULL);                      // Last time written
    if (status != ERROR_SUCCESS)
        {
        SetLastError(status);
HACKassert("Error on RegQueryInfoKey",status == ERROR_SUCCESS);
        throw;                          // XXX Should be a different exception.
        }
```

```
#else
    dwMaxNameLen = 4096;
#endif
    char * extensionNameBuffer = new char [dwMaxNameLen +2];
    CString openAppEntry;
    CString tmp;

// Scan until we get to a key starting with a  "." ...

    DWORD dwIndex;
    for (dwIndex = 0; ; dwIndex++)
        {
        DWORD dwNameLen = dwMaxNameLen;
        status = RegEnumKeyEx(
                    HKEY_CLASSES_ROOT,   // Key to enumerate
                    dwIndex,             // index of subkey to enumerate
                    extensionNameBuffer,// buffer for subkey name
                    &dwNameLen,          // size   of  subkey name buffer
                    NULL,                // reserved.
                    NULL,                // buffer for class name
                    NULL,                // size   of  class name
                    NULL);               // Last time written to
        if ((status != ERROR_NO_MORE_ITEMS) && (status != ERROR_SUCCESS))
            {
            SetLastError(status);
HACKassert("Error on RegEnumKeyEx",status == ERROR_SUCCESS);
            }
        if (status != ERROR_SUCCESS)     // All done ...
            break;

        if (extensionNameBuffer[0] != '.')
            continue;        // Doesn't start with "."

// OK, get the app name and check it ...
        if (CheckDocumentOpenAssociation(extensionNameBuffer,openAppEntry))
            {                            // Add to our list.
            tmp.Format("%s=%s", extensionNameBuffer,
                    (const char *) AtpAttrib::QuoteString(openAppEntry));
            associationList.ParseLine(tmp);
            }

        status = ERROR_SUCCESS; // Just a place to set a breakpoint.
        }

    delete [] extensionNameBuffer;

}



#ifdef  SOURCE_CONTROL_BLOCK
Project Name: E-PS Client/Server program
E-PS Inc.
Copyright 1996, 1997. All Rights Reserved.
SUBSYSTEM: Utility utility classes and methods for client/server project
$Workfile: SystemInventory.cpp $
$Author: Jrward $
$Revision: 12 $
$Date: 1/06/97 11:06a $
$Modtime: 1/05/97 7:37p $
```

```
$History: SystemInventory.cpp $
*
* *****************  Version 12  *****************
* User: Jrward       Date: 1/06/97    Time: 11:06a
* Updated in $/atp/Utility
* Removed our configuration data from "system inventory": we access it
* separately now.
*
* *****************  Version 9  *****************
* User: Jrward       Date: 1/02/97    Time: 10:59a
* Updated in $/atp/Utility
* Added SOURCE_CONTROL_BLOCK stuff.

// JRW 961203   GetSystemInventory moved here from utility.cpp.
// JRW 961204   GetAllDocumentOpenAssociations added,
//              non-Win95 stuff conditioned out in GetSystemInventory.
// JRW 961205   Added waveOutGetNumDevs, etc. for sound output.
#endif    // SOURCE_CONTROL_BLOCK
```

```
// atpsock.cpp : implementation of the CAtpSocket class

#include <stdlib.h>
#include "stdafx.h"
#include <assert.h>
#include <memory.h>

#include "atpsock.h"
#include "atpexcept.h"
#include "atpdataval.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#define DEBUG_OUTPUT

#define DEFAULT_TIMEOUT          150
#define DEFAULT_BLOCKSIZE        16384
#define CRYPT_RANDOM_LEN     32
#define CRYPT_SERVER_KEYHOLDER   "ATP_SERVER_KEYHOLDER"
#define CRYPT_KEYHOLDER_NAME(username)   (username + "_atp_keyholder")
#define DEFAULT_PROVIDER_NAME    "ElectronicParcelService"

////////////////////////////
// AtpSocketImplementations

AtpSocket::AtpSocket(IAtpCallback * pCB) {
  m_pCB = pCB;  // JRW 961127
  m_buflen = 0;
  m_timeout = DEFAULT_TIMEOUT;
  m_block_pref = DEFAULT_BLOCKSIZE;

  m_fast = FALSE;
  m_encrypt_data = TRUE;
  m_really_encrypt = FALSE;
  m_provider = NULL;
  m_outgoing_session_key = NULL;
  m_incoming_session_key = NULL;
}

AtpSocket::~AtpSocket() {
  if (m_outgoing_session_key) delete m_outgoing_session_key;
  if (m_incoming_session_key) delete m_incoming_session_key;
  if (m_provider) delete m_provider;
}

void AtpSocket::Close() {
  if (m_hSocket)
    CAsyncSocket::Close();
  else
    closesocket(my_hSocket);
}

CString AtpSocket::GetUserName() {
  return m_username;
}
```

```
CString AtpSocket::GetProviderName() {
  return m_providerName;
}

SocketTraffic& AtpSocket::GetSocketTraffic() {
  return traffic;
}

// Initialize the ATP client protocol: Read the initial response from server
// throws ProtocolErrorException

void AtpSocket::InitializeClient() {
  unsigned long opt = 0;
  int code;
  AtpAttrib arg;

  GetPeerName(m_peerhost, m_peerport);
  my_hSocket = Detach();

  ioctlsocket(my_hSocket, FIONBIO, &opt); // set blocking

  RecvResponse(code, arg);
  if (code != ATP_RSP_SERVER_READY)
    throw AtpProtocolException();
  m_providerName = arg["ProviderName"];
}


void AtpSocket::InitializeServer() {
  unsigned long opt = 0;
  AtpAttrib arg;
  GetPeerName(m_peerhost, m_peerport);
  my_hSocket = Detach();
  ioctlsocket(my_hSocket, FIONBIO, &opt); // set blocking
  arg["ProviderName"] = DEFAULT_PROVIDER_NAME;
  SendResponse(ATP_RSP_SERVER_READY, arg);
}


#ifdef USE_CRYPT_LIBRARY
void AtpSocket::MangleData(CryptProvider& provider, CByteArray& data,
                CryptKey& pwd_key) {
  // Mangle the data
  pwd_key.EncryptData(data, TRUE);
  CryptHash data_hash(provider);
  data_hash.HashData(data);
  data_hash.GetValue(data);
  pwd_key.EncryptData(data, TRUE);
}
#endif

// Authentication & encryption solution:
// For efficiency reasons, there are two scenarious possibly used:
// One using encryption and one sending the data in the open
//
// A. Authentication only
//    The sequence of commands is the following:
//    1. the client creates random data sequence and sends it to the server
//    2. the server encodes that sequence using the user's passwd and
//       send back the hash (also encoded). It also sends back its own
```

```
//       random data sequence
//    3. the client checks the correctness of the received hash; then it
//       encrypts server's sequence and returns its hash (also encoded)
//    4. the server verifies the correctness of the received hash
//
// B. Authentication and encryption
//    We have a private key (password) and would like to exchange
//    our session keys by using it. Unfortunately, the Microsoft encryption
//    provider can encrypt a session key only with an RSA public/private
//    key pair. As a result, the steps from the above process are
//    modified to reflect this:
//    1. the client creates an RSA key pair, and sends the server its
//       public key and a random data sequence.
//    2. the server creates an RSA key pair, and sends the client its
//       public key. It also encodes the received sequence using the
//       user's password and sends back the hash (also encoded) as well
//       as its session key (encrypted with the client's public key)
//    3. the client checks the correctness of the received hash; then it
//       encrypts server's session key blob and sends back the hash
//       (also encoded) as well as its own session key (encrypted with
//       server's public key)
//    4. the server verifies the correctness of the received hash
//

BOOL AtpSocket::AuthenticateClient(CString username, CString password) {
  int code;
  AtpAttrib arg;

  m_username = username;

#ifdef USE_CRYPT_LIBRARY
  CryptKey *rsa = NULL;
  CryptKey *server_rsa = NULL;
  CryptKey *outgoing_session_key = NULL;
  CryptKey *incoming_session_key = NULL;
  CByteArray random, received, key_export;

  try {
    // Initialize the crypto api
    m_provider = new CryptProvider(CRYPT_KEYHOLDER_NAME(username));
  } catch (CMyException ex) {
    ex;
    m_provider = NULL;
  }
#endif

  if (!m_provider) {
    // simple authentication -- cryptography won't work
    arg.RemoveAll();
    arg[ATP_ARG_USER] = username;
    arg[ATP_ARG_PASS] = password;
    SendCommand(ATP_CMD_LOGIN, arg);
    RecvResponse(code, arg);
    if (code != ATP_RSP_LOGIN_OK)
      throw AtpProtocolException();
    return TRUE;
  }

#ifdef USE_CRYPT_LIBRARY
  try {
```

```
//////////////
// Phase 1

// Generate random data
CryptProvider& provider = *m_provider;
provider.GenerateRandom(random, CRYPT_RANDOM_LEN);

// Generate RSA key and a session key if the data will be encrypted
if (m_encrypt_data) {
  try {
rsa = new CryptKey(provider, AT_KEYEXCHANGE);
rsa->ExportKey(key_export, PUBLICKEYBLOB);
// use a stream encryption algorithm for the data
outgoing_session_key = new CryptKey(provider, CALG_RC4);
  } catch (CryptAPIException ex) {
ex;
m_encrypt_data = FALSE;
if (rsa) delete rsa;
rsa = NULL;
if (outgoing_session_key) delete outgoing_session_key;
outgoing_session_key = NULL;
  }
}

// Send data (phase 1)
arg.RemoveAll();
arg[ATP_ARG_USER] = username;
arg[ATP_ARG_SALT] = AtpAttrib::UnparseBinary(random);
if (m_encrypt_data)
  arg[ATP_ARG_RSA] = AtpAttrib::UnparseBinary(key_export);
SendCommand(ATP_CMD_LOGIN, arg);

//////////////
// Phase 2

// Create key based on the password
CryptHash pwd_hash(provider);
pwd_hash.HashData(password);
CryptKey pwd_key(provider, CALG_RC2, pwd_hash);

// Calculate expected result
MangleData(provider, random, pwd_key);

// Get a response and handle it
RecvResponse(code, arg);

switch (code) {
case ATP_RSP_LOGIN_ENCRYPTION:
  if (!m_encrypt_data)
throw AtpProtocolException();
  break;
case ATP_RSP_LOGIN_NO_ENCRYPTION:
  if (m_encrypt_data) {
m_encrypt_data = FALSE;
delete rsa;
rsa = NULL;
  }
  break;
case ATP_RSP_LOGIN_FAIL:
  if (rsa) delete rsa;
```

```
    return FALSE;
  default:
    throw AtpProtocolException();
  }

  // check if the server knows the password
  try {
    BOOL correct = TRUE;
    AtpAttrib::ParseBinary(arg[ATP_ARG_RESULT], received);
    if (received.GetSize() != random.GetSize())
  correct = FALSE;
    else
  for (int i = 0; i < received.GetSize(); i++)
    if (received[i] != random[i]) {
      correct = FALSE;
      break;
    }
    if (!correct) {
  if (rsa) delete rsa;
  if (server_rsa) delete server_rsa;
  if (incoming_session_key) delete incoming_session_key;
  if (outgoing_session_key) delete outgoing_session_key;
  return FALSE;
    }
  } catch (FormatException ex) {
    ex;
    throw AtpProtocolException();
  }

  // Get server's public key
  if (m_encrypt_data) {
    try {
  AtpAttrib::ParseBinary(arg[ATP_ARG_RSA], received);
    } catch (FormatException ex) {
  ex;
  throw AtpProtocolException();
    }
    try {
  server_rsa = new CryptKey(provider, received);
    } catch (CryptAPIException ex) {
  ex;
  throw AtpProtocolException();
    }
  }

  // Get server's random data/session key
  try {
    AtpAttrib::ParseBinary(arg[ATP_ARG_SALT], received);
  } catch (FormatException ex) {
    ex;
    throw AtpProtocolException();
  }

  if (m_encrypt_data) {
    try {
  incoming_session_key = new CryptKey(provider, received);
    } catch (CryptAPIException ex) {
  ex;
  throw AtpProtocolException();
    }
```

```
    }

    //////////////
    // Phase 3

    // Mangle server's random data
    MangleData(provider, received, pwd_key);

    if (m_encrypt_data) {
      // Export the client session key using server's rsa
      outgoing_session_key->ExportKey(key_export, SIMPLEBLOB, *server_rsa);
    }

    // Send data (phase 3)
    arg.RemoveAll();
    arg[ATP_ARG_USER] = username;
    arg[ATP_ARG_RESULT] = AtpAttrib::UnparseBinary(received);
    if (m_encrypt_data)
      arg[ATP_ARG_SALT] = AtpAttrib::UnparseBinary(key_export);
    SendCommand(ATP_CMD_LOGIN2, arg);

    delete rsa;
    rsa = NULL;
    delete server_rsa;
    server_rsa = NULL;

    //////////////
    // Phase 4

    RecvResponse(code, arg);
    if (code != ATP_RSP_LOGIN_OK)
      throw AtpProtocolException();

  } catch (CMyException ex) {
    ex;
    if (rsa) delete rsa;
    if (server_rsa) delete server_rsa;
    if (outgoing_session_key) delete outgoing_session_key;
    if (incoming_session_key) delete incoming_session_key;
    throw;
  }

  m_outgoing_session_key = outgoing_session_key;
  m_incoming_session_key = incoming_session_key;
  m_really_encrypt = m_encrypt_data;
#endif

  return TRUE;
}

// to be implemented
BOOL AtpSocket::AuthenticateServer() {
  // implement a private key encryption
  // scheme a la Kerberos (passwords do not
  // travel in the open)

  AtpCommand cmd;
  AtpAttrib arg;

  //////////////
```

```
// Phase 1

RecvCommand(cmd, arg);
if (cmd != ATP_CMD_LOGIN) {
  arg.RemoveAll();
  SendResponse(ATP_RSP_LOGIN_EXPECTED, arg);
  return FALSE;
}

m_username = arg[ATP_ARG_USER];
CString password = ServerObtainPassword(m_username);
if (password.IsEmpty()) {
  SendResponse(ATP_RSP_LOGIN_FAIL, arg);
  return FALSE;
}

if (arg[ATP_ARG_PASS] == password) {
  // All is good, authentication successful (simple one)
  arg.RemoveAll();
  SendResponse(ATP_RSP_LOGIN_OK, arg);
  return TRUE;
}

#ifdef USE_CRYPT_LIBRARY
  CryptKey *rsa = NULL;
  CryptKey *client_rsa = NULL;
  CryptKey *outgoing_session_key = NULL;
  CryptKey *incoming_session_key = NULL;

  try {
    // Initialize crypto api
    CByteArray random, received, key_export;
    m_provider = new CryptProvider(CRYPT_SERVER_KEYHOLDER);
    CryptProvider& provider = *m_provider;

    //////////////
    // Phase 2

    // Create key from password
    CryptHash pwd_hash(provider);
    pwd_hash.HashData(password);
    CryptKey pwd_key(provider, CALG_RC2, pwd_hash);

    // Get client's data and mangle it
    try {
      AtpAttrib::ParseBinary(arg[ATP_ARG_SALT], received);
    } catch (FormatException ex) {
      ex;
      throw AtpProtocolException();
    }
    MangleData(provider, received, pwd_key);

    // Get client's rsa key if encrypting
    if (m_encrypt_data) {
      try {
    AtpAttrib::ParseBinary(arg[ATP_ARG_RSA], key_export);
    client_rsa = new CryptKey(provider, key_export);
      } catch (FormatException ex) {
    ex;
    throw AtpProtocolException();
```

```
  } catch (CryptAPIException ex) {
ex;
m_encrypt_data = FALSE;
  }
}

// Create rsa key and session key if encrypting
if (m_encrypt_data) {
  try {
rsa = new CryptKey(provider, AT_KEYEXCHANGE);
rsa->ExportKey(key_export, PUBLICKEYBLOB);
// use a stream encryption algorithm for the data
outgoing_session_key = new CryptKey(provider, CALG_RC4);
outgoing_session_key->ExportKey(random, SIMPLEBLOB, *client_rsa);
  } catch (CryptAPIException ex) {
ex;
m_encrypt_data = FALSE;
if (client_rsa) delete rsa;
client_rsa = NULL;
if (rsa) delete rsa;
rsa = NULL;
if (outgoing_session_key) delete outgoing_session_key;
outgoing_session_key = NULL;
  }
}

// if not encrypting data, just send a random string
if (!m_encrypt_data)
  provider.GenerateRandom(random, CRYPT_RANDOM_LEN);

// Send data (phase 2)
arg.RemoveAll();
arg[ATP_ARG_SALT] = AtpAttrib::UnparseBinary(random);
arg[ATP_ARG_RESULT] = AtpAttrib::UnparseBinary(received);
if (m_encrypt_data) {
  arg[ATP_ARG_RSA] = AtpAttrib::UnparseBinary(key_export);
  SendResponse(ATP_RSP_LOGIN_ENCRYPTION, arg);
}
else
  SendResponse(ATP_RSP_LOGIN_NO_ENCRYPTION, arg);

//////////////
// Phase 3

// Calculate expected result
MangleData(provider, random, pwd_key);

// Get a response and handle it
RecvCommand(cmd, arg);
if (cmd != ATP_CMD_LOGIN2) {
  arg.RemoveAll();
  SendResponse(ATP_RSP_LOGIN_EXPECTED, arg);
  throw AtpProtocolException();
}

// Verify if the client knows the password
try {
  BOOL correct = TRUE;
  AtpAttrib::ParseBinary(arg[ATP_ARG_RESULT], received);
  if (received.GetSize() != random.GetSize())
```

```
  correct = FALSE;
    else
  for (int i = 0; i < received.GetSize(); i++)
    if (received[i] != random[i]) {
      correct = FALSE;
      break;
    }
    if (!correct) {
  if (rsa) delete rsa;
  if (client_rsa) delete client_rsa;
  if (incoming_session_key) delete incoming_session_key;
  if (outgoing_session_key) delete outgoing_session_key;
  return FALSE;
    }
  } catch (FormatException ex) {
    ex;
    throw AtpProtocolException();
  }

  // Get client's session key
  try {
    AtpAttrib::ParseBinary(arg[ATP_ARG_SALT], received);
  } catch (FormatException ex) {
    ex;
    throw AtpProtocolException();
  }

  if (m_encrypt_data) {
    try {
  incoming_session_key = new CryptKey(provider, received);
    } catch (CryptAPIException ex) {
  ex;
  throw AtpProtocolException();
    }
  }

  delete rsa;
  rsa = NULL;
  delete client_rsa;
  client_rsa = NULL;

  //////////////
  // Phase 4

  // All is good, authentication successful
  arg.RemoveAll();
  SendResponse(ATP_RSP_LOGIN_OK, arg);

} catch (CMyException ex) {
  ex;
  if (rsa) delete rsa;
  if (client_rsa) delete client_rsa;
  if (incoming_session_key) delete incoming_session_key;
  if (outgoing_session_key) delete outgoing_session_key;
  throw;
}

m_incoming_session_key = incoming_session_key;
m_outgoing_session_key = outgoing_session_key;
m_really_encrypt = m_encrypt_data;
```

```
#endif

  return TRUE;
}

CString AtpSocket::ServerObtainPassword(CString username) {
  username;
  return "";
}

void AtpSocket::ServerLoop() {
  AtpCommand cmd;
  AtpAttrib arg;

  while(TRUE) {
    RecvCommand(cmd, arg);
    switch(cmd) {

    case ATP_CMD_BLOCK:
      long pref;
      try {
    pref = AtpAttrib::ParseInt(arg[ATP_ARG_PREF]);
      } catch (FormatException ex) {
    ex;
        throw AtpProtocolException();
      }
      ServerNegotiateBlock(pref);
      break;

    case ATP_CMD_SEND:
      ServerHandleSend(arg);
      break;

    case ATP_CMD_RECV:
      ServerHandleRecv(arg);
      break;

    case ATP_CMD_CLOSE:
      return;

    default:
      throw AtpProtocolException();
    }
  }
}

///// Client operations follow /////

// Negotiate a block size with the server
// throws AtpException
void AtpSocket::NegotiateBlock(long preferred) {
  int code;
  AtpAttrib arg;

  while (TRUE) {
    arg.RemoveAll();
    arg[ATP_ARG_PREF] = AtpAttrib::UnparseInt(preferred);
    SendCommand(ATP_CMD_BLOCK, arg);

    RecvResponse(code, arg);
```

```
    switch (code) {
    case ATP_RSP_BLOCK_OK:
      m_block_pref = preferred;
      return;
    case ATP_RSP_BLOCK_BAD:
    case ATP_RSP_BLOCK_PRESET:
      // we are mellow: agree with the suggested size
      try {
    preferred = AtpAttrib::ParseInt(arg[ATP_ARG_PREF]);
      } catch (FormatException ex) {
        ex;
    throw AtpProtocolException();
      }
      break;
    default:
      throw AtpProtocolException();
    }
  }
}

// virtual, to be overridden if necessary
void AtpSocket::ServerNegotiateBlock(long client_preferred) {
  // always agree by default
  ServerBlockAgree(client_preferred);
  m_block_pref = client_preferred;
}

void AtpSocket::ServerBlockAgree(long preferred) {
  AtpAttrib arg;
  arg[ATP_ARG_PREF] = AtpAttrib::UnparseInt(preferred);
  SendResponse(ATP_RSP_BLOCK_OK, arg);
}

void AtpSocket::ServerBlockDisagree(long preferred, BOOL permanent) {
  AtpAttrib arg;
  arg[ATP_ARG_PREF] = AtpAttrib::UnparseInt(preferred);
  if (permanent)
    SendResponse(ATP_RSP_BLOCK_PRESET, arg);
  else
    SendResponse(ATP_RSP_BLOCK_BAD, arg);
}


// Send a file to the server. Returns TRUE if file is accepted
// throws AtpException
BOOL AtpSocket::SendFile(AtpFile& fl) {
  int code;
  AtpAttrib arg;
  long from, len;
  CTime start;
  CTimeSpan interval;

  // send file info
  fl.GetAttributes(arg);
  SendCommand(ATP_CMD_SEND, arg);

  while (TRUE) {
    // get a response/request & handle it
    RecvResponse(code, arg);
    switch(code) {
```

```
case ATP_RSP_SEND_REJECTED:
  return FALSE;

case ATP_RSP_STALL:
  long stall;
  try { stall = AtpAttrib::ParseInt(arg[ATP_ARG_STALL]); }
  catch (FormatException ex) {
    ex;
    throw AtpProtocolException();
  }
  RemoteStall(stall);        // do not time out for that time
  RecvResponse(code, arg);           // ++ check this
  arg.RemoveAll();
  SendCommand(ATP_CMD_READY, arg);
  continue;

case ATP_RSP_SEND_DONE:
  // the transfer is complete
  return TRUE;

case ATP_RSP_SEND_DATA:
  break;

default:
  throw AtpProtocolException();
}

// data block has been requested
try {
  from = AtpAttrib::ParseInt(arg[ATP_ARG_FROM]);
  len = AtpAttrib::ParseInt(arg[ATP_ARG_LEN]);
}
catch (FormatException ex) {
  ex;
  throw AtpProtocolException();
}

// check if we can allow a large block transfer
long stall = m_pCB ? m_pCB->RequestTransfer(len) : 0;
if (stall > 0) {
  arg.RemoveAll();
  arg[ATP_ARG_STALL] = AtpAttrib::UnparseInt(stall);
  SendCommand(ATP_CMD_STALL, arg);
  LocalStall(stall);
  arg.RemoveAll();
  SendCommand(ATP_CMD_STALLEND, arg);
  continue;
}
else if (stall < 0) {
  // hopeless, close connection
  arg.RemoveAll();
  SendCommand(ATP_CMD_CLOSE, arg);
  ShutDown(2);
  throw AtpInterruptException();
}

// send the block
arg.RemoveAll();
arg[ATP_ARG_FROM] = AtpAttrib::UnparseInt(from);
```

```
    arg[ATP_ARG_LEN] = AtpAttrib::UnparseInt(len);
    SendCommand(ATP_CMD_DATA, arg);
    SendData(fl, from, len);
  }

  return TRUE;
}

// virtual controller function.
// Override this to define the behaviour at Send
void AtpSocket::ServerHandleSend(AtpAttrib& arg) {
  // Do not accept files in the default implementation
  arg;
  ServerRejectSend();
}

void AtpSocket::ServerRejectSend() {
  AtpAttrib arg;
  SendResponse(ATP_RSP_SEND_REJECTED, arg);
}

void AtpSocket::ServerAcceptSend(AtpFile& fl) {
  AtpCommand cmd;
  AtpAttrib arg;
  long from, len;

  while (!fl.IsComplete()) {
    // get a vacant block
    fl.RequestBlock(m_block_pref, from, len);

    // can we download a large block?
    long stall = m_pCB ? m_pCB->RequestTransfer(len) : 0;
    if (stall > 0) {
      arg.RemoveAll();
      arg[ATP_ARG_STALL] = AtpAttrib::UnparseInt(stall);
      SendResponse(ATP_RSP_STALL, arg);
      LocalStall(stall);
      arg.RemoveAll();
      SendResponse(ATP_RSP_STALLEND, arg);
      RecvCommand(cmd, arg);     // ++ verify if this is ready

      fl.ClearRequest(from, len);
      continue;
    }
    else if (stall < 0) {
      // hopeless, close connection
      arg.RemoveAll();
      SendResponse(ATP_RSP_CLOSE, arg);
      ShutDown(2);
      throw AtpInterruptException();
    }

    // send request for the vacant block
    arg.RemoveAll();
    arg[ATP_ARG_FROM] = AtpAttrib::UnparseInt(from);
    arg[ATP_ARG_LEN] = AtpAttrib::UnparseInt(len);
    SendResponse(ATP_RSP_SEND_DATA, arg);

    // get a command and handle it
    RecvCommand(cmd, arg);
```

%-12345X@PJL SET PAGEPROTECT=AUTO
@PJL SET RESOLUTION=600
@PJL ENTER LANGUAGE=PCL

```
// IDFServer.cpp : Implementation of the IDFServer and IDFile classes
// C++ code file
// (c) 1996 ACS
//

#include "stdafx.h"
#include "assert.h"
#include "atpfile.h"
#include "atpdata.h"
#include "atpexcept.h"
#include "utility.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


AtpFile::AtpFile() {
  m_fbound = FALSE;
  m_tentative = TRUE;
  m_fl = NULL;
  m_fsize = 0;
  m_floaded = 0;
}

AtpFile::AtpFile(CFile *fl) {
  m_fbound = FALSE;
  m_tentative = TRUE;
  m_fl = NULL;
  m_fsize = 0;
  m_floaded = 0;
  Associate(fl);
}

AtpFile::~AtpFile() {
}

CFile* AtpFile::GetFile() {
  return m_fl;
}

void AtpFile::AssignAttributes(CFile* fl, AtpAttrib& attrib) {
  try {
    m_fsize = m_floaded = fl->GetLength();
    CFileStatus stat;
    fl->GetStatus(stat);
    attrib[ATP_ARG_NAME] = fl->GetFileName();
    attrib[ATP_ARG_SIZE] = AtpAttrib::UnparseInt(m_fsize);
    attrib[ATP_ARG_TIME] = AtpAttrib::UnparseTime(stat.m_mtime);
    attrib[ATP_ARG_CHKSUM] = AtpAttrib::UnparseInt(CalculateChecksum(fl));
  } catch (CFileException *ex) {
    ex->Delete();
    throw FileErrorException();
  }
}

void AtpFile::Associate(CFile* fl) {
  assert(!m_fbound);
```

```
  AtpAttrib attrib;
  AssignAttributes(fl, attrib);

  m_attribs = attrib;
  m_fl = fl;
  m_fbound = TRUE;
}

long AtpFile::CalculateChecksum(CFile* fl) const {
  fl;
  return 0;
}

void AtpFile::BindToFile(CFile* fl, long loaded) {
  assert(!m_fbound);
  //assert(m_fsize > 0);

  if (m_fsize < 0)
    throw InvalidOperationException();
  m_floaded = loaded;
  try {
    fl->SetLength(m_fsize);
  } catch (CFileException *ex) {
    ex->Delete();
    throw FileErrorException();
  }
  m_fl = fl;
  m_fbound = TRUE;
}

void AtpFile::SetAttributes(const AtpAttrib& attrib) {
  AtpAttrib attribs = attrib;
  long new_size = 0;
  if (!attribs[ATP_ARG_SIZE].IsEmpty())
    try {
      new_size = AtpAttrib::ParseInt(attribs[ATP_ARG_SIZE]);
      if (m_fbound && new_size != m_fsize) {
        //throw InvalidOperationException();
        m_fl->SetLength(new_size);
      }
      m_tentative = FALSE;
    } catch (FormatException ex) {
      ex;
      HACKassert("Invalid size format", TRUE);
      new_size = 0;
    }
  else
    m_tentative = TRUE;

  m_fsize = new_size;
  m_attribs = attrib;
}

void AtpFile::GetAttributes(AtpAttrib& attrib) {
  attrib = m_attribs;
}

long AtpFile::GetSize() const {
  return m_fsize;
```

```
}

long AtpFile::GetLoadedSize() const {
  return m_floaded;
}

BOOL AtpFile::IsComplete() const {
  return (!m_tentative && m_floaded == m_fsize);
}

BOOL AtpFile::IsBound() const {
  return m_fbound;
}

BOOL AtpFile::RequestBlock(long pref, long& from, long& len) {
  assert(m_fbound);

  if (!m_fbound)
    throw InvalidOperationException();

  if (IsComplete()) return FALSE;

  from = m_floaded;
  len = (pref > 0) ? pref : m_fsize - from;
  if (from + len > m_fsize)
    len = m_fsize - from;
  return TRUE;
}

BOOL AtpFile::ClearRequest(long from, long len) {
  from; len;
  return TRUE;
}

void AtpFile::WriteBlock(void *data, long from, long len) {
  assert(m_fbound);

  if (!m_fbound)
    throw InvalidOperationException();
  if (from < 0 || len < 0 || from + len > m_fsize) {
    HACKassert("Invalid range", TRUE);
    throw RangeErrorException();
  }

  // do not write again (or when file is complete)
  if (from + len <= m_floaded) return;

  try {
    m_fl->Seek(from, CFile::begin);
    m_fl->Write(data, len);
    if (from <= m_floaded && from + len > m_floaded)
      m_floaded = from + len;
  } catch (CFileException *ex) {
    ex->Delete();
    throw FileErrorException();
  }
}

void AtpFile::ReadBlock(void *data, long from, long len) {
  assert(m_fbound);
```

```
  assert(from >= 0 && len > 0);

  if (!m_fbound)
    throw InvalidOperationException();
  if (from < 0 || len < 0 || from + len > m_fsize)
    throw RangeErrorException();

  try {
    m_fl->Seek(from, CFile::begin);
    if ((long)m_fl->Read(data, len) != len)
      throw FileErrorException();
  } catch (CFileException *ex) {
    ex->Delete();
    throw FileErrorException();
  }
}

BOOL AtpFile::operator==(const AtpFile& comp) const {
  try {
    AtpAttrib attrib1, attrib2;
    attrib1 = m_attribs;
    attrib2 = comp.m_attribs;
    return (attrib1[ATP_ARG_NAME] == attrib2[ATP_ARG_NAME] &&
        AtpAttrib::ParseInt(attrib1[ATP_ARG_SIZE]) ==
        AtpAttrib::ParseInt(attrib2[ATP_ARG_SIZE]) &&
        AtpAttrib::ParseInt(attrib1[ATP_ARG_CHKSUM]) ==
        AtpAttrib::ParseInt(attrib2[ATP_ARG_CHKSUM]) &&
    AtpAttrib::ParseTime(attrib1[ATP_ARG_TIME]) ==
    AtpAttrib::ParseTime(attrib2[ATP_ARG_TIME]));
  } catch (FormatException ex) {
    ex;
    return FALSE;
  }
}


// End of code

#ifdef  SOURCE_CONTROL_BLOCK
 Project Name: E-PS Client/Server program
 E-PS Inc.
 Copyright 1996, 1997. All Rights Reserved.
 SUBSYSTEM: EPS Protocol
 $Workfile: atpfile.cpp $
 $Author: Ttonchev $
 $Revision: 9 $
 $Date: 1/31/97 9:58a $
 $Modtime: 1/31/97 9:15a $

 $History: atpfile.cpp $
 *
 * *****************  Version 9  *****************
 * User: Ttonchev     Date: 1/31/97    Time: 9:58a
 * Updated in $/atp/Protocol
 * fixed problems with IsComplete() when the default size is 0
 *
 * *****************  Version 8  *****************
 * User: Ttonchev     Date: 1/28/97    Time: 1:23p
 * Updated in $/atp/Protocol
```

```
 *
 * *****************  Version 7  *****************
 * User: Jrward       Date: 1/27/97    Time: 12:07p
 * Updated in $/atp/Protocol
 * Added VSS history stuff.
#endif    // SOURCE_CONTROL_BLOCK
```

5

```
//
// C++ header file
// (c) 1996 ACS
//

#ifndef _ATPFILE_H_
#define _ATPFILE_H_

#include "stdafx.h"
#include "atpattrib.h"

class AtpFile : public CObject {
private:
  // disallow copy contructor and assignment
  AtpFile(const AtpFile&);
  void operator=(const AtpFile&);

public:
  AtpFile();
  AtpFile(CFile *fl);        // performs Associate on fl. Does not adopt fl.
  virtual ~AtpFile();

public:

  // Methods for interaction with the class
  BOOL operator==(const AtpFile& comp) const;

  virtual void SetAttributes(const AtpAttrib& attrib);
  virtual void GetAttributes(AtpAttrib& attrib);

  virtual BOOL IsComplete() const;
  virtual BOOL IsBound() const;
  virtual CFile* GetFile();

  virtual long GetSize() const;
  virtual long GetLoadedSize() const;

  virtual BOOL RequestBlock(long pref, long& from, long& len);
  virtual BOOL ClearRequest(long from, long len);

  virtual void WriteBlock(void *data, long from, long len);
  virtual void ReadBlock(void *data, long from, long len);

protected:

  // interaction with the subclasses
  void BindToFile(CFile* fl, long loaded = 0);  // bind to a new file

  virtual void AssignAttributes(CFile* fl, AtpAttrib& attrib);
  void Associate(CFile* fl);     // associate with an existing file

private:
  long CalculateChecksum(CFile *fl) const;


  // Member Variables Section

private:
  // these member variables can be accessed by using the methods
  // please do not use directly
```

```
  CFile *m_fl;
  AtpAttrib m_attribs;
  BOOL m_fbound, m_tentative;
  long m_fsize, m_floaded;
};


#endif // _ATPFILE_H_

// End of headers

#ifdef  SOURCE_CONTROL_BLOCK
 Project Name: E-PS Client/Server program
 E-PS Inc.
 Copyright 1996, 1997. All Rights Reserved.
 SUBSYSTEM: E-parcel.com
 $Workfile: atpfile.h $
 $Author: Ttonchev $
 $Revision: 4 $
 $Date: 1/31/97 9:58a $
 $Modtime: 1/31/97 9:09a $

 $History: atpfile.h $
 *
 * *****************  Version 4  *****************
 * User: Ttonchev     Date: 1/31/97    Time: 9:58a
 * Updated in $/atp/Include
 * fixed problems with IsComplete() when the default size is 0
 *
 * *****************  Version 3  *****************
 * User: Jrward       Date: 1/27/97    Time: 12:13p
 * Updated in $/atp/Include
 * Added VSS history stuff.
#endif    // SOURCE_CONTROL_BLOCK
```

```
// atpclient.cpp : implementation of the AtpClientSocket class

#include "stdafx.h"
#include <stdlib.h>
#include <assert.h>

#include "atpclient.h"
#include "atpexcept.h"
#include "atpdata.h"
#include "atpmemfile.h"
#include "SystemInventory.h"

AtpClientSocket::AtpClientSocket(IDFServer *idf_server, IAtpCallback * pCB)
   : AtpSocket(pCB)     // JRW 961127
{
  m_idf_server = idf_server;
  m_pPCB = NULL;
}

AtpClientSocket::~AtpClientSocket() {
}

void AtpClientSocket::Run(AtpAttrib& providers, long bsize) {
  try {
    InitializeClient();
    conn.provider.ParseLine(providers[GetProviderName()]);
    if (!AuthenticateClient(conn.provider["UserName"], conn.provider["UserPassword"])) {
      ClientQuit();
      throw AtpAuthenticationException();
    }
    NegotiateBlock(bsize);
    ClientLoop();
    ClientQuit();
  } catch (CMyException ex) {
    ex;
    DWORD err = GetLastError();
    Close();
    throw;
  }
}

void AtpClientSocket::RunAnon(IDFile *idf, long bsize) {
  try {
    InitializeClient();
    NegotiateBlock(bsize);
    ClientLoopAnon(idf);
    ClientQuit();
  } catch (CMyException ex) {
    ex;
    DWORD err = GetLastError();
    Close();
    throw;
  }
}

void AtpClientSocket::InitObserver(AtpConnUID uid,
                                   IAtpProgressCallback *pPCB) {
  m_pPCB = pPCB;
  conn.uid = uid;
  conn.state = AtpConnection::CONNECTING;
```

```
  GetPeerName(conn.peer_address, conn.peer_port);
  conn.traffic = &GetSocketTraffic();
  conn.idf = NULL;
  if (m_pPCB) m_pPCB->ProcessConnectionState(conn);
}

void AtpClientSocket::ClientLoop() {
  // Set state to WAITING
  conn.state = AtpConnection::WAITING;
  conn.peer_address = m_peerhost;
  conn.peer_port = m_peerport;
  if (m_pPCB) m_pPCB->ProcessConnectionState(conn);

  // Request current registration information
  {
    AtpAttrib arg;
    arg[ATP_ARG_NAME] = ATP_FILE_REGISTRATION;
    arg[ATP_ARG_SPECIAL] = AtpAttrib::UnparseBool(TRUE);

    AtpMemFile afl(arg);
    if (RecvFile(afl)) {
      arg.RemoveAll();
      arg[ATP_ARG_NAME] = ATP_FILE_BENCHMARK;
      arg[ATP_ARG_SPECIAL] = AtpAttrib::UnparseBool(TRUE);
      AtpMemFile afl2(arg);
      RecvFile(afl2);
    }
    else {
    }
  }

  // Send current configuration
  {
    AtpAttrib arg, inv;
    arg[ATP_ARG_NAME] = ATP_FILE_CONFIG;
    arg[ATP_ARG_SPECIAL] = AtpAttrib::UnparseBool(TRUE);

    AtpMemFile afl(arg);
    CFile *fl = afl.GetFile();
    CArchive ar(fl, CArchive::store);
    GetSystemInventory(inv);
    inv.UnparseFile(ar);
    ar.Flush();
    fl->Flush();

    SendFile(afl);
  }

  // Request file list
  {
    AtpAttrib arg;
    arg[ATP_ARG_NAME] = ATP_FILE_LIST;
    arg[ATP_ARG_SPECIAL] = AtpAttrib::UnparseBool(TRUE);

    AtpMemFile afl(arg);
    if (!RecvFile(afl)) return;

    // ok, got it, now update local file database
    CFile *fl = afl.GetFile();
    fl->SeekToBegin();
```

```
    CArchive arch(fl, CArchive::load);

    CString line;
    while (arch.ReadString(line)) {
      arg.RemoveAll();
      arg.ParseLine(line);
      IDFile *idf = m_idf_server->FindIDFile(arg);
      if (!idf) {
        idf = new IDFile(arg);
        try {
          m_idf_server->Add(idf);
        } catch (FileErrorException ex) {
          ex;
          continue;
        }
      }
    }
  }

  // Download incomplete files
  for (int i = 0; i < m_idf_server->GetSize(); i++) {
    IDFile *idf = m_idf_server->GetIDFile(i);
    // Lock checks if a transfer is already in progress
    CSingleLock lock(idf->GetLock());
    if (!idf->IsComplete() && lock.Lock(0)) {
      BOOL ok;
      try {
        ok = RecvFile(*idf, this);
      } catch (AtpException ex) {
        ex;
        idf->Store();
        lock.Unlock();
        throw;
      }
      lock.Unlock();
      if (!ok) {
        // ++ delete this idf file -- no longer available
        continue;
      }
      else {
        // File received. Set state to WAITING.
        conn.state = AtpConnection::WAITING;
        if (m_pPCB) m_pPCB->ProcessConnectionState(conn);
      }
    }
    if (idf->IsComplete() && !idf->IsFinalized())
      idf->Finalize();
  }
}

void AtpClientSocket::ClientLoopAnon(IDFile *idf) {
  conn.state = AtpConnection::WAITING;
  conn.peer_address = m_peerhost;
  conn.peer_port = m_peerport;
  if (m_pPCB) m_pPCB->ProcessConnectionState(conn);

  // Get requested file
  {
    CSingleLock lock(idf->GetLock());
    // Lock checks if a transfer is already in progress
```

```
    if (!idf->IsComplete() && lock.Lock(0)) {
      BOOL ok;
      try {
        ok = RecvFile(*idf, this);
      } catch (AtpException ex) {
        ex;
        idf->Store();
        lock.Unlock();
        throw;
      }
      lock.Unlock();
      if (!ok) {
        // ++ delete this idf file -- no longer available
        return;
      }
      else {
        // File received. Set state to WAITING.
        conn.state = AtpConnection::WAITING;
        if (m_pPCB) m_pPCB->ProcessConnectionState(conn);
      }
    }
    if (idf->IsComplete() && !idf->IsFinalized())
      idf->Finalize();
  }
}

void AtpClientSocket::ProcessWrittenBlock(AtpFile *atp) {
  conn.state = AtpConnection::RECEIVING;
  conn.idf = (IDFile *) atp;
  if (m_pPCB) m_pPCB->ProcessConnectionState(conn);
}

void AtpClientSocket::ProcessStall(AtpFile *atp) {
  conn.state = AtpConnection::PAUSED;
  conn.idf = (IDFile *) atp;
  if (m_pPCB) m_pPCB->ProcessConnectionState(conn);
}

void AtpClientSocket::Close() {
  CSocket::Close();
  conn.state = AtpConnection::CLOSING;
  if (m_pPCB) m_pPCB->ProcessConnectionState(conn);
}

///// Debugging utilities /////

#ifdef _DEBUG
void AtpClientSocket::AssertValid() const
{
  AtpSocket::AssertValid();
}

void AtpClientSocket::Dump(CDumpContext& dc) const
{
  AtpSocket::Dump(dc);
}
#endif //_DEBUG


#ifdef SOURCE_CONTROL_BLOCK
```

```
Project Name: E-PS Client/Server program
E-PS Inc.
Copyright 1996, 1997. All Rights Reserved.
SUBSYSTEM: ClientProtocol
$Workfile: atpclient.cpp $
$Author: Ttonchev $
$Revision: 12 $
$Date: 1/31/97 2:49p $
$Modtime: 1/31/97 2:38p $

$History: atpclient.cpp $
*
* *****************  Version 12  *****************
* User: Ttonchev     Date: 1/31/97    Time: 2:49p
* Updated in $/atp/ClientProtocol
* Fixed locking mechanism
*
* *****************  Version 11  *****************
* User: Ttonchev     Date: 1/31/97    Time: 1:03p
* Updated in $/atp/ClientProtocol
*
* *****************  Version 10  *****************
* User: Jrward       Date: 1/27/97    Time: 12:08p
* Updated in $/atp/ClientProtocol
* Added VSS history stuff.
#endif    // SOURCE_CONTROL_BLOCK
```

```
// atpclient.h : interface of the AtpClientSocket class

#ifndef __ATPCLIENT_H__
#define __ATPCLIENT_H__

#include "stdafx.h"
#include "atpsock.h"
#include "callback.h"
#include "idfserver.h"

typedef long AtpConnUID;

class AtpConnection {
public:
  enum AtpState {
    CONNECTING,          // Connection just created
    WAITING,             // Connection established; waiting for transfer
    RECEIVING,           // Transfering data
    PAUSED,            // Transfer paused for some reason
    CLOSING            // Connection will be closed
   };

   AtpConnUID uid;
   AtpState state;
   AtpAttrib provider;
   CString peer_address;
   unsigned int peer_port;
   SocketTraffic *traffic;
   IDFile *idf;            // Valid only if RECEIVING, or PAUSED
 };


 class AtpClientSocket : public AtpSocket, public IAtpRecvCallback {
   // Inherited relevant public methods:
   // InitializeClient(), AuthenticateClient()

 public:
   AtpClientSocket(IDFServer *idf_server, IAtpCallback * pCB = NULL);
   virtual ~AtpClientSocket();

   void InitObserver(AtpConnUID uid, IAtpProgressCallback *pPCB);
   AtpConnection* GetConnectionState();

   void Run(AtpAttrib& provider, long bsize);
   void RunAnon(IDFile *idf, long bsize);

   virtual void ProcessWrittenBlock(AtpFile *atp);
   virtual void ProcessStall(AtpFile *atp);

   virtual void Close();

  private:
    IDFServer *m_idf_server;
    IAtpProgressCallback *m_pPCB;
    AtpConnection conn;

    void ClientLoop();
    void ClientLoopAnon(IDFile *idf);

   protected:
```

```
#ifdef _DEBUG
   void AssertValid() const;
   void Dump(CDumpContext& dc) const;
#endif // _DEBUG

};

#endif // __ATPCLIENT_H__



#ifdef  SOURCE_CONTROL_BLOCK
 Project Name: E-PS Client/Server program
 E-PS Inc.
 Copyright 1996, 1997. All Rights Reserved.
 SUBSYSTEM: E-parcel.com
 $Workfile: atpclient.h $
 $Author: Jrward $
 $Revision: 5 $
 $Date: 1/27/97 12:13p $
 $Modtime: 1/27/97 12:10p $

 $History: atpclient.h $
 *
 * *****************  Version 5  *****************
 * User: Jrward       Date: 1/27/97    Time: 12:13p
 * Updated in $/atp/Include
 * Added VSS history stuff.
#endif    // SOURCE_CONTROL_BLOCK
```

```
//
// C++ code file
// (c) 1996 ACS
//
// JRW 961216 Added m_configVals, so we can pass in block xfer size, etc.

#include "stdafx.h"
#include "objbase.h"

#include <assert.h>

#include "SocketControl.h"
#include "AtpAttrib.h"
#include "atpexcept.h"
#include "sockutil.h"
#include "utility.h"
#include "atpdata.h"

extern int WriteErrorLog(const char *, const char *);

SocketControl::SocketControl(AtpAttrib & configVals) :
    m_configVals(configVals)
{
  m_requestValue = 0;    // JRW 961127 Default is, you can transfer anything.
  m_lsock = NULL;
  m_defaultThreadPriority = THREAD_PRIORITY_NORMAL;
  uid_max = 0;
}

SocketControl::~SocketControl() {
  if (m_lsock) delete m_lsock;
  m_lsock = NULL;
  for (int i = 0; i < m_threads.GetSize(); i++)
    delete (CWinThread *) m_threads[i];
}

BOOL SocketControl::Initialize(IDFServer *idfs, IAtpProgressCallback *pPCB) {
  m_idfs = idfs;
  m_pPCB = pPCB;
  m_lsock = new CListeningSocket(this);
  if (m_lsock->Create(ATP_PORT) && m_lsock->Listen())
    return TRUE;
  return FALSE;
}

void SocketControl::ProcessPendingAccept(CAsyncSocket *sock) {
  //for (int i = 0; i < m_todelete.GetSize(); i++)
  //  delete m_todelete[i];
  //m_todelete.RemoveAll();

  CAsyncSocket *atp_sock = new CAsyncSocket();
  if (!sock->Accept(*atp_sock)) {
    delete atp_sock;
    return;
  }

  ThreadInfo *info = new ThreadInfo();
  info->me = this;
  info->sock = atp_sock->Detach();
  delete atp_sock;
```

```
  StartSocketThread(info);
}

void SocketControl::CloseConnections() {
  // close all connections
  CSingleLock lock(&m_sockmutex);
  lock.Lock();
  for (int i = 0; i < m_sockets.GetSize(); i++)
    // shutdown the communication
    // this cannot be done cleanly -- CSocket is not thread-safe
    shutdown(((CSocket *) m_sockets[i])->m_hSocket, 2);

  lock.Unlock();

  // wait for all threads to terminate
  while (TRUE) {
    CSingleLock lock(&m_mutex);
    lock.Lock();
    int num = m_conns.GetSize();
    lock.Unlock();
    if (num == 0) break;
    Sleep(10);
  }
}

void SocketControl::PollServers(AtpAttrib providers) {
  AtpAttrib provider;
  CString s_provider, s_params;
  POSITION pos = providers.GetStartPosition();
  while (pos) {
    providers.GetNextAssoc(pos, s_provider, s_params);
    provider.RemoveAll();
    provider.ParseLine(s_params);

    if (ServerConnectionAlready(provider))
      continue;

    BOOL bSuccessful;
    bSuccessful = ConnectToServer(provider);

    // Write something to the log ...
    CTime t = CTime::GetCurrentTime();
    CString stime = t.Format("%#c");
    int status = ERROR_SUCCESS;
    CString tmp;
    tmp.Format("%s attempt at %s -- %s",
               (bSuccessful ? "Successful" : "Unsuccessful"),
               (const char *) stime,
               (const char *) provider["ServerName"]);
    status = WriteErrorLog("LastServerPoll", tmp);
  }

  //Important:
  //  LockIDFList() does lock the list and may cause other threads to block.
  //  For that reason, it is important to lock it only for a while. Also,
  //  operations related to the IDFServer may thus cause a deadlock.
  //  Therefore, it is best to use the trick below in cases where one needs
  //  extensive computation or IDFServer related operations.
```

```
  CPtrArray updates;
  CPtrArray *idfs = m_idfs->LockIDFList();
  for (int i = 0; i < idfs->GetSize(); i++) {
    AtpAttrib arg;
    IDFile *idf = (IDFile *) (*idfs)[i];
    idf->GetAttributes(arg);
    if (!arg["SERVERNAME"].IsEmpty() && !idf->IsLocked())
      updates.Add(idf);
  }
  m_idfs->UnlockIDFList();

  for (int j = 0; j < updates.GetSize(); j++)
    ConnectToServerAnon((IDFile*)updates[j]);
}

// See if we already have a connection going for a particular
// server before the client starts a (new) connection to it ...
// Note: not really thread safe, the server could initiate a connection
// in the interval between when we check and when we start, but
// Theo wants it this way.

BOOL SocketControl::ServerConnectionAlready(AtpAttrib provider)
{

// XXX Note:
//  ... need to tell Theo, we have to convert serverName to a dotted-name
// IP address for this to work, since GetPeerName only returns dotted-name
// IP addresses, not the local.host.com-style name we may be looking for.
// Also, there may be more than one local.host.com-style name for
// a given server.
// Also, certain types of cluster technologies for servers work by
// mapping a given local.host.com to different IP addresses in a round-robin
// fashion, etc.
// Ergo, it sounds like we need our own identification mechanism, this
// stuff about relying on either the IP address or the domain name
// is a temporary hack.

    const CPtrArray * p_conns = GetAllConnectionStates();
assert(("Bad internal pointer p_conns",p_conns != NULL));
    int connectionCount = p_conns->GetSize();
    for (int idx = 0; idx < connectionCount; idx++)
      {
    /*
        CString thisServerName =
                    ((AtpConnection*)(p_conns->GetAt(idx)))->peer_address;
    // see if ip address
    if (inet_addr(serverName) != INADDR_NONE &&
        serverName == thisServerName)
      return TRUE;
    // see if real name
    hostent *info = gethostbyname(serverName);
    for (int i = 0; info->h_addr_list[i]; i++) {
      in_addr addr;
      addr.S_un.S_addr = *(long*)info->h_addr_list[i];
      if (inet_ntoa(addr) == thisServerName)
            return TRUE;
    }
    */
    if (provider["ProviderName"] ==
```

```
        ((AtpConnection*)(p_conns->GetAt(idx)))->provider["ProviderName"])
      return TRUE;
      }

    return FALSE;
}


BOOL SocketControl::ConnectToServer(AtpAttrib provider) {
  ThreadInfo *info = new ThreadInfo();
  info->me = this;
  info->sock = INVALID_SOCKET;
  info->host = provider["ServerName"];
  info->port = ATP_SERVER_PORT;
  info->provider = provider;
  info->idf = NULL;

  StartSocketThread(info);

  return TRUE;
}

BOOL SocketControl::ConnectToServerAnon(IDFile *idf) {
  AtpAttrib arg;
  idf->GetAttributes(arg);
  CString server = arg[ATP_ARG_SERVER];
  UINT port;
  try {
    port = AtpAttrib::ParseInt(arg[ATP_ARG_PORT]);
  } catch (FormatException ex) {
    ex;
    port = ATP_ANON_SERVER_PORT;
  }

  ThreadInfo *info = new ThreadInfo();
  info->me = this;
  info->sock = INVALID_SOCKET;
  info->host = server;
  info->port = port;
  info->idf = idf;

  StartSocketThread(info);

  return TRUE;
}

void SocketControl::StartSocketThread(LPVOID pParam) {
  CWinThread *thread = AfxBeginThread(InitSocketThread, pParam,
                      m_defaultThreadPriority, 0,
                      CREATE_SUSPENDED);
  thread->m_bAutoDelete = FALSE;
  m_threads.Add(thread);
  thread->ResumeThread();
}

UINT SocketControl::InitSocketThread(LPVOID pParam) {
  ThreadInfo *info = (ThreadInfo *) pParam;

  if (info->sock == INVALID_SOCKET)
    info->sock = SocketConnectTo(info->host, info->port);
```

```
  if (info->sock == INVALID_SOCKET)
    return 1;

  AtpClientSocket csock(info->me->m_idfs, info->me);
  assert(csock.Attach(info->sock));
  info->me->ModifySocketList(csock, TRUE);
  info->me->RunClientSocket(csock, info->provider, info->idf);
  info->me->ModifySocketList(csock, FALSE);
  delete info;

  return 0;
}

void SocketControl::RunClientSocket(AtpClientSocket& csock, AtpAttrib& provider,
                                    IDFile *idf) {
  CSingleLock lock(&m_mutex);
  lock.Lock();
  int new_uid = uid_max++;
  lock.Unlock();
  csock.InitObserver(new_uid, this);
  try {
    if (idf) {
      csock.RunAnon(idf, AtpAttrib::ParseInt(m_configVals["ProtocolBlockSize"]));
    }
    else {
      AtpAttrib providers;
      providers.ParseLine(m_configVals["Providers"]);
      csock.Run(providers, AtpAttrib::ParseInt(m_configVals["ProtocolBlockSize"]));
    }
  } catch (AtpAuthenticationException ex) {
    ex;
    AfxMessageBox("Cannot authenticate to server");
  }
  catch (CMyException ex)
  {
    ex;
  }
}

void SocketControl::ModifySocketList(CSocket& sock, BOOL add) {
  CSingleLock lock(&m_sockmutex);
  lock.Lock();
  if (add)
    m_sockets.Add(&sock);
  else
    for (int i = 0; i < m_sockets.GetSize(); i++)
      if (m_sockets[i] == &sock) {
        m_sockets.RemoveAt(i);
    i--;
      }
  lock.Unlock();
}

// JRW: Return state reference for a given connection:
// IN: uid: id of the connection
// OUT: conn: AtpConnection reference

BOOL SocketControl::GetConnectionState(long uid, AtpConnection*& conn) {
  CSingleLock lock(&m_mutex);
  lock.Lock();
```

```
  BOOL found = FALSE;
  int size = m_conns.GetSize();
  for (int i = 0; i < size; i++) {
    AtpConnection *mconn = (AtpConnection *) m_conns[i];
    if (mconn->uid == uid) {
      conn = mconn;
      found = TRUE;
      break;
    }
  }
  lock.Unlock();
  return found;
}

// JRW: Return pointer to array of all connections.

const CPtrArray* SocketControl::GetAllConnectionStates() {
  return &m_conns;
}

SocketTraffic SocketControl::GetSocketTraffic() {
  CSingleLock lock(&m_mutex);
  lock.Lock();

  SocketTraffic traffic = m_oldtraffic;
  int size = m_conns.GetSize();
  for (int i = 0; i < size; i++)
    traffic += *((AtpConnection *) m_conns[i])->traffic;

  lock.Unlock();
  return traffic;
}

void SocketControl::ProcessConnectionState(const AtpConnection& conn) {
  if (conn.state == AtpConnection::CONNECTING ||
      conn.state == AtpConnection::CLOSING) {
    CSingleLock lock(&m_mutex);
    lock.Lock();
    switch (conn.state) {
    case AtpConnection::CONNECTING:
      m_conns.Add((void *)&conn);
      break;
    case AtpConnection::CLOSING:
      for (int i = 0; i < m_conns.GetSize(); i++)
    if (((AtpConnection*)m_conns[i])->uid == conn.uid) {
      m_conns.RemoveAt(i);
      m_oldtraffic += *conn.traffic;
      break;
    }
      break;
    }
    lock.Unlock();
  }

  if (m_pPCB) m_pPCB->ProcessConnectionState(conn);
}

// JRW 961127
// This implementation uses a Set/Get model for implementing
```

```
// RequestTransfer, rather than a callback function --
// for the moment, it looks like we don't need the generality,
// and this is dumb and easy.

long SocketControl::RequestTransfer(long size)
{
    (VOID *) size;
    return m_requestValue;
}

void SocketControl::SetRequestTransfer(long requestValue)
{
    m_requestValue = requestValue;
}


void SocketControl::SetProtocolThreadPriority(int iPriority)     // Set it to this.
{
  m_defaultThreadPriority = iPriority;

  for (int i = 0; i < m_threads.GetSize(); i++) {
    DWORD retval;
    CWinThread *thread = (CWinThread *)m_threads[i];
    // see if still active
    if (!GetExitCodeThread(thread->m_hThread, &retval) ||
    retval != STILL_ACTIVE) {
      m_threads.RemoveAt(i);
      delete thread;
      i--;
      continue;
    }
    thread->SetThreadPriority(iPriority);
  }
}

int SocketControl::GetThreadPriority() // Read from OS.
{
  return m_defaultThreadPriority;    // Read from OS, not this..
}


// End of code

#ifdef  SOURCE_CONTROL_BLOCK
 Project Name: E-PS Client/Server program
 E-PS Inc.
 Copyright 1996, 1997. All Rights Reserved.
 SUBSYSTEM: ClientProtocol
 $Workfile: SocketControl.cpp $
 $Author: Ttonchev $
 $Revision: 17 $
 $Date: 2/03/97 1:03p $
 $Modtime: 2/03/97 1:03p $

 $History: SocketControl.cpp $
 *
 * *****************  Version 17  *****************
 * User: Ttonchev     Date: 2/03/97    Time: 1:03p
 * Updated in $/atp/ClientProtocol
```

```
 * added some comments
 *
 * *****************  Version 16  *****************
 * User: Ttonchev       Date: 1/31/97    Time: 2:50p
 * Updated in $/atp/ClientProtocol
 * made socket connection asynchronous, so that blocking of the main
 * thread is avoided
 *
 * *****************  Version 15  *****************
 * User: Ttonchev       Date: 1/31/97    Time: 1:04p
 * Updated in $/atp/ClientProtocol
 * Added restart of Smartload connections
 *
 * *****************  Version 14  *****************
 * User: Jrward         Date: 1/27/97    Time: 12:08p
 * Updated in $/atp/ClientProtocol
 * Added VSS history stuff.
#endif    // SOURCE_CONTROL_BLOCK
```

```
//
// C++ header file
// (c) 1996 ACS
//

// JRW 961216   m_configVals added, so we can pass xfer block size.

#ifndef _SOCKETCONTROL_H_
#define _SOCKETCONTROL_H_

#include "stdafx.h"
#include "afxmt.h"
#include "callback.h"
#include "lstnsock.h"
#include "atpsock.h"
#include "idfserver.h"
#include "atpclient.h"
#include "AtpAttrib.h"

const ATP_PORT = 608;
const ATP_SERVER_PORT = 609;
const ATP_ANON_SERVER_PORT = 610;


class SocketControl : public ISockCallback, public IAtpCallback,
                public IAtpStateCallback, public IAtpProgressCallback
{
public:
    SocketControl(AtpAttrib & configVals) ;
    virtual ~SocketControl();

public:
  virtual void ProcessPendingAccept(CAsyncSocket *sock);
  //virtual void ProcessClose(CAsyncSocket *sock);

  virtual BOOL GetConnectionState(long uid, AtpConnection*& conn);
  virtual const CPtrArray* GetAllConnectionStates();
  virtual SocketTraffic GetSocketTraffic();

  BOOL Initialize(IDFServer *idfs, IAtpProgressCallback *pPCB);
  void PollServers(AtpAttrib providers);
  BOOL ConnectToServerAnon(IDFile *idf);
  BOOL ConnectToServer(AtpAttrib provider);
  void CloseConnections();

  virtual void ProcessConnectionState(const AtpConnection& conn);

private:
  struct ThreadInfo {
    SocketControl* me;
    SOCKET sock;
    CString host;
    UINT port;
    AtpAttrib provider;
    IDFile *idf;
  };

  void StartSocketThread(LPVOID pParam);
  static UINT InitSocketThread(LPVOID pParam);
  void RunClientSocket(AtpClientSocket& sock, AtpAttrib& provider, IDFile *idf);
```

```
  void ModifySocketList(CSocket& sock, BOOL add);

// JRW 961127
// This is the dumb and simple implementation, without really
// using the generality of a callback.
public:
  virtual long RequestTransfer(long size);
  virtual void SetRequestTransfer(long requestValue);

  virtual void SetProtocolThreadPriority(int iPriority); // Set it to this.
  virtual int  GetThreadPriority();              // Read from OS.


private:
  BOOL ServerConnectionAlready(AtpAttrib provider);

private:
  AtpAttrib & m_configVals;   // JRW Reference to global settings.

  IDFServer *m_idfs;
  IAtpProgressCallback *m_pPCB;

  CListeningSocket *m_lsock;
  CPtrArray m_conns;

  AtpConnUID uid_max;
  SocketTraffic m_oldtraffic;

  CMutex m_mutex, m_sockmutex;

  CPtrArray m_threads, m_sockets;

  // JRW 961127
  long m_requestValue;
  int  m_defaultThreadPriority;
};


#endif // _SOCKETCONTROL_H_

// End of headers




#ifdef  SOURCE_CONTROL_BLOCK
 Project Name: E-PS Client/Server program
 E-PS Inc.
 Copyright 1996, 1997. All Rights Reserved.
 SUBSYSTEM: E-parcel.com
 $Workfile: SocketControl.h $
 $Author: Ttonchev $
 $Revision: 13 $
 $Date: 1/31/97 2:50p $
 $Modtime: 1/31/97 2:19p $

 $History: SocketControl.h $
 *
 * *****************  Version 13  *****************
```

```
 * User: Ttonchev      Date: 1/31/97    Time: 2:50p
 * Updated in $/atp/Include
 * made socket connection asynchronous, so that blocking of the main
 * thread is avoided
 *
 * *****************  Version 12  *****************
 * User: Ttonchev      Date: 1/31/97    Time: 1:04p
 * Updated in $/atp/Include
 * Added restart of Smartload connections
 *
 * *****************  Version 11  *****************
 * User: Jrward        Date: 1/30/97    Time: 12:42p
 * Updated in $/atp/Include
 * changed port for SL server to 610 (from 609)
 *
 * *****************  Version 10  *****************
 * User: Jrward        Date: 1/27/97    Time: 12:13p
 * Updated in $/atp/Include
 * Added VSS history stuff.
#endif    // SOURCE_CONTROL_BLOCK
```

```
// ClientApp.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "stdlib.h"
#include <assert.h>

#include "winnls.h"

#include "utility.h"
#include "atpattrib.h"
#include "atpexcept.h"

#include "ClientApp.h"
#include "ClientAppDlg.h"
#include "SmartLoad.h"
#include "packfile.h"
#include "DbgConfigDlg.h"
#include "TextDisplayDlg.h"

#include "PropertyPageDialAutomatically.h"
#include "PropertyPageFirewall.h"
#include "PropertyPageInternetConnection.h"
#include "PropertyPageIPAssignment.h"
#include "PropertyPageProxyInfo.h"
#include "PropertyPageRasScript.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// So that different modules will report the same build date ...
CString g_ClientAppBuildDate;

// For debugging and testing only ...
BOOL    g_bSimulateTransfer     = FALSE;
BOOL    g_bSampleDisabled       = FALSE;

HINSTANCE CClient::m_hRasApi32Dll;

/////////////////////////////////////////////////////////////////////////////
// CClient

BEGIN_MESSAGE_MAP(CClient, CWinApp)
  //{{AFX_MSG_MAP(CClient)
  // NOTE - the ClassWizard will add and remove mapping macros here.
  //    DO NOT EDIT what you see in these blocks of generated code!
  //}}AFX_MSG
  ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP();

/////////////////////////////////////////////////////////////////////////////
// CClient construction

CClient::CClient():
    m_bRASAlreadyActive(FALSE),
    m_bSplashScreen(TRUE)
{
```

```
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance

// So that different modules will report the same build date ...
// (Initialize here, not statically, so that we don't get a bogus
// memory-leak message from the debugger.)

    g_ClientAppBuildDate = "(Beta of " __DATE__  " " __TIME__ ".)";

// Get RasApi32.dll, if not exists, RAS was not installed so that
// Ras connection can not be made.
    if(NULL == m_hRasApi32Dll)
        m_hRasApi32Dll = LoadLibrary("rasapi32.dll");

// This is just for convenience tracking down memory leaks using
// the MS VC++ debugger:
#ifdef  _DEBUG
#include <crtdbg.h>
    long int dbgAlloc = -1;
    _CrtSetBreakAlloc(dbgAlloc);
#endif  // _DEBUG
}

////////////////////////////////////////////////////////////////////////////
// The one and only CClient object

CClient theApp;

// Server to connect to (temporary)
#define DEFAULT_ATP_PROVIDER "ElectronicParcelService"
#define DEFAULT_ATP_SERVER   "theserver.e-parcel.com" // "199.103.208.230"  // Theo's machi
ne via terra.net
#define DEFAULT_ATP_PASSWORD "testing"

// Registry key names we use:
// Note:
// Probably at least some of our values should go into HKEY_CURRENT_USER:
// .... have to figure this out later.
// Note:
// Normally these should be set up by InstallShield install/uninstall.
// ... Key1, Key2 are for full deletion: Windows95 deletes subkeys
// automatically, but the documentation for WindowsNT says you can't.

#define COMPANY "EPARCEL"          // Must be what InstallShield uses for
#define APPNAME "Client"           // <Company> <AppName> <Version>
#define VERSION "970128"
const char * g_Client_Registration_Key =
                                {"Software\\" COMPANY "\\" APPNAME "\\" VERSION};
const char * Registration_Key1={"Software\\" COMPANY "\\" APPNAME};
const char * Registration_Key2={"Software\\" COMPANY};

// Returns ERROR_SUCCESS on success ...

int WriteErrorLogValue(const char * ItemName, const char * resultValue)
{
HACKassert("Bad ItemName pointer passed", ItemName != NULL);
HACKassert("Bad resultValue pointer passed", resultValue != NULL);

    HKEY    hKey = NULL;
    DWORD   keyDisposition;
```

```
    LONG     status;
    status = RegCreateKeyEx(
        HKEY_LOCAL_MACHINE,                          // open key
        g_Client_Registration_Key,                   // subkey
        0,                                           // reserved
        NULL,                                        // address of class string
        REG_OPTION_NON_VOLATILE,                     // options flag
        KEY_ALL_ACCESS,                              // security access
        NULL,                                        // key security structure
        &hKey,                                       // opened handle
        &keyDisposition);                            // disposition value
    if (status != ERROR_SUCCESS)
        {
        SetLastError(status);
HACKassert("Error on RegCreateKeyEx",status == ERROR_SUCCESS);
        return status;
        }
    if ((keyDisposition != REG_CREATED_NEW_KEY) &&
        (keyDisposition != REG_OPENED_EXISTING_KEY))
        {
        status = ERROR_INVALID_DATA;
        SetLastError(status);
HACKassert("Bad disposition code on RegCreateKeyEx",status == ERROR_SUCCESS);
        return status;
        }

    CString quotedValue = AtpAttrib::QuoteString(resultValue);
    status  = RegSetValueEx(
        hKey,                                        // open key
        (char *)(const char *)ItemName,              // which item
        NULL,                                        // reserved
        REG_SZ,                                      // type
        (CONST BYTE *)(const char *)quotedValue,     // dataValue
        strlen(quotedValue)+1);                      // buffer size needed
    if (status != ERROR_SUCCESS)
        {
        SetLastError(status);
HACKassert("Cannot set registry value",status == ERROR_SUCCESS);
        return status;
        }

    status = RegCloseKey(hKey);
    hKey = NULL;
    if (status != ERROR_SUCCESS)
        {
        SetLastError(status);
HACKassert("Cannot close hKey handle",status == ERROR_SUCCESS);
        return status;
        }
    return status;
}

int WriteErrorLog(const char * ItemName, const char * resultValue)
{
    CTime currentTime = CTime::GetCurrentTime();
    CString stime = currentTime.Format("%#c");
    int status = WriteErrorLogValue("LastErrorLogTime", stime);
    if (status != ERROR_SUCCESS)
        return status;
    return WriteErrorLogValue(ItemName, resultValue);
```

```
}

// Show usage in a message box
// Note: Should put strings into resource file ...

VOID CClient::ShowHowToUse
    (const char * msg, const char * BadCommandLinePart)
{
HACKassert("Bad msg pointer passed", msg != NULL);
HACKassert("Bad BadCommandLinePart passed", BadCommandLinePart != NULL);
    CString message;
    if (CString(BadCommandLinePart) == "")
        message = "RTFM, Jocko ...\n" + CString(msg) + "\n";
    else{
        message = CString("Bad command line:\n   Try again!\n   ")
                    + CString(msg)
                    + CString("\n   Bad part: ")
                    + CString(BadCommandLinePart)
                    + CString("\n");
        }
    AtpAttrib defaultValues = GetFactoryDefaultValues();
    CString tmp = defaultValues.UnparseLine("\n");
    message +=
            "Special use:\n"
            "\t/LoadRequest\n"
            "\t@<filename>\n"
            "\t/AskConfiguration\n"
            "DEBUGGING USE ONLY:\n"
            "\t/SimulateTransfer\n"
            "\t/SampleDisabled\n"
            "\t/ShowValues\n"
            "\t/WriteValues\n"
            "\t/ResetValues\n"
            "\t/ClearValues\n"
            "\t/NoSplash\n"
            "\t/DebugBreak\n"
            "\t/Quit\n"
            "Factory defaults:\n" +
            tmp +
            "\n"
            "Hope this helps!";

// Note: maybe no application window, so just pass "NULL".
    ::MessageBox(NULL,(char *) (const char *) message, "Usage instructions ...",
                        MB_OK | MB_ICONEXCLAMATION);
}

BOOL CClient::ValidBoolean(CString & sBoolean)
{
if ((sBoolean == "TRUE") ||
    (sBoolean == "YES"))
    return TRUE;
if ((sBoolean == "FALSE") ||
    (sBoolean == "NO"))
    return TRUE;
return FALSE;
}

// Insanity checks on RAS entry names: return FALSE if looks bogus.
```

```
BOOL CClient::ValidRASScriptName(CString & RASScriptName)
{
if (AtpAttrib::ParseBool(m_configVals["ConnectionPermanent"]))
    return TRUE;     // Direct connection, not dialup.
if (!AtpAttrib::ParseBool(m_configVals["DialAutomatically"]))
    return TRUE;     // No auto-dial, won't be dialing up ourselves.

// XXX Need to validate against actual RAS entries ...
int idx = RASScriptName.GetLength();
if (idx < 2)
    return FALSE;
if (idx > 24)
    return FALSE;
return TRUE;
}

// Validity checks on serial numbers: return FALSE if looks bogus.
// XXX Note: serial number of "0" will be to indicate we do registration on
// first connection to client (not implemented yet).

BOOL CClient::ValidSerial(CString & serialNumber)
{
if (serialNumber == "0")
    return TRUE;

int idx = serialNumber.GetLength();
if (idx < 2)
    return FALSE;

// Secret "checksum": first two digits and last two digits must be "42".
// (With apologies to Douglass P. Adams)

if (serialNumber[0] != '4')
    return FALSE;
if (serialNumber[1] != '2')
    return FALSE;
if (serialNumber[idx-2] != '4')
    return FALSE;
if (serialNumber[idx-1] != '2')
    return FALSE;

if (idx > 24)            // Too long.
    return FALSE;

for (idx--; idx >= 0; idx--)
    {
    if (!isdigit(serialNumber[idx]))
        return FALSE;
    }

return TRUE;
}

// Return FALSE if it looks bogus ...

BOOL CClient::ValidSampleTime(CString & sampleTime)
{
int idx = sampleTime.GetLength();
if (idx < 1)
    return FALSE;
```

```
for (idx--; idx >= 0; idx--)
    {
    if (!isdigit(sampleTime[idx]))
        return FALSE;
    }
int nMilliseconds = -1;
sscanf ((const char *) sampleTime, "%u", &nMilliseconds);

if (nMilliseconds < 1000)
    return (FALSE);
if (nMilliseconds > 60000)
    return (FALSE);

return TRUE;
}

BOOL CClient::ValidClientPollingPeriod(CString & clientPollingPeriod)
{
int idx = clientPollingPeriod.GetLength();
if (idx < 1)
    return FALSE;

for (idx--; idx >= 0; idx--)
    {
    if (!isdigit(clientPollingPeriod[idx]))
        return FALSE;
    }
int nSeconds = -1;
sscanf ((const char *) clientPollingPeriod, "%u", &nSeconds);

if (nSeconds == 0)
    return TRUE;              // Special case, no client polling.
if (nSeconds == 15)
    return TRUE;              // Special case for debugging and testing ...
if (nSeconds < 10*60)         // Minimum, 10 minutes.
    return (FALSE);
if (nSeconds > 12*60*60)      // Maximum, 12 hours.
    return (FALSE);

return TRUE;
}

BOOL CClient::ValidSampleCount(CString & sampleCount)
{
int idx = sampleCount.GetLength();
if (idx < 1)
    return FALSE;

for (idx--; idx >= 0; idx--)
    {
    if (!isdigit(sampleCount[idx]))
        return FALSE;
    }
int nPeriods = -1;
sscanf ((const char *) sampleCount, "%u", &nPeriods);

if (nPeriods < 3)
    return (FALSE);
if (nPeriods > 15)
```

```
    return (FALSE);

return TRUE;
}

// Insanity checks on phone numbers: return FALSE if looks bogus.
// NOTE: We don't always need a phone number, which is why we need
// the default here ...

BOOL CClient::ValidRASPhoneNumber(CString & phoneNumber)
{

int idx = phoneNumber.GetLength();
if (idx < 2)
    return FALSE;
if (idx > 24)
    return FALSE;
for (idx--; idx >= 0; idx--)
    {
    if (!isdigit(phoneNumber[idx]))
        return FALSE;
    }
return TRUE;
}

BOOL CClient::InvalidConfigurationValue(const char * name)
{
    ShowHowToUse("Invalid configuration value ...",
        CString(name) + "=" + m_configVals[name]);
    return FALSE;
}

VOID CClient::SetSerialInstallationDate(AtpAttrib & configVals, CTime & t,
        CString & textString)
{
    configVals["SerialInstallYear"]     = AtpAttrib::UnparseInt(t.GetYear());
    configVals["SerialInstallMonth"]    = AtpAttrib::UnparseInt(t.GetMonth());
    configVals["SerialInstallDay"]      = AtpAttrib::UnparseInt(t.GetDay());
    configVals["SerialInstallHour"]     = AtpAttrib::UnparseInt(t.GetHour());
    configVals["SerialInstallMinute"]   = AtpAttrib::UnparseInt(t.GetMinute());
    configVals["SerialInstallSecond"]   = AtpAttrib::UnparseInt(t.GetSecond());

    textString  =   configVals["SerialInstallYear"]     + "-" +
                    configVals["SerialInstallMonth"]    + "-" +
                    configVals["SerialInstallDay"]      + " " +
                    configVals["SerialInstallHour"]     + ":" +
                    configVals["SerialInstallMinute"]   + ":" +
                    configVals["SerialInstallSecond"];

}

VOID CClient::GetSerialInstallationDate(AtpAttrib & configVals, CTime & t,
        CString & textString)
{
    t = CTime(
            AtpAttrib::ParseInt(configVals["SerialInstallYear"]),
            AtpAttrib::ParseInt(configVals["SerialInstallMonth"]),
            AtpAttrib::ParseInt(configVals["SerialInstallDay"]),
            AtpAttrib::ParseInt(configVals["SerialInstallHour"]),
            AtpAttrib::ParseInt(configVals["SerialInstallMinute"]),
            AtpAttrib::ParseInt(configVals["SerialInstallSecond"]));
```

```
    textString  =   configVals["SerialInstallYear"]    + "-" +
                    configVals["SerialInstallMonth"]   + "-" +
                    configVals["SerialInstallDay"]     + " " +
                    configVals["SerialInstallHour"]    + ":" +
                    configVals["SerialInstallMinute"]  + ":" +
                    configVals["SerialInstallSecond"];
}

BOOL CClient::DoCommandLine()
{
// Pick up some defaults ...
     m_configVals = GetFactoryDefaultValues();
     CString tmp = m_configVals.UnparseLine();  // TEMP DEBUG

// Read saved values from registry, if any.
     try {
          m_configVals.ParseRegistry(g_Client_Registration_Key);
          }
     catch (FormatException ex)
          {   // error while reading the registry. ignore for now
HACKassert("Error reading configuration values from registry: you may continue",FALSE);
          ex;
          }
      tmp = m_configVals.UnparseLine();  // TEMP DEBUG

// For debugging in the field, keep track of what's up ...
     {
     int status = ERROR_SUCCESS;
     status = WriteErrorLog("LastCommandLine", m_lpCmdLine);
HACKassert("BadValue",(status==ERROR_SUCCESS)||(status==ERROR_FILE_NOT_FOUND));
     }

// OK, now that we have gotten the current configuration values, we
// can parse the command line -- there may be some special flags that override
// the defaults, so we had to read the default values first.

     CString CommandLine = m_lpCmdLine;

// Check command line ...
// (Note: probably should use CWinApp::ParseCommandLine,
// not this horse-hockey ...

     CString parseChar;  // First character, to check for special flags.
     CString token;      // First item on command line.
     int idx     = 0;
     int inext   = 0;
     for (;;)
          {
          CommandLine.TrimLeft();
          CommandLine.TrimRight();
          token = "";
// Empty line ...
          if (CommandLine == "")
               break;      // All done.

// XXX Not smart about embedded spaces yet ...
// Should scan until quote parity is zero, and first whitespace.

          inext = CommandLine.FindOneOf(" \t\r\n");
```

```
        if (inext == -1)
            {
            token = CommandLine;     // ... the last bit of the command line
            CommandLine = "";
            }
        else{
            token = CommandLine.Mid(0,inext);
            CommandLine = CommandLine.Mid(inext);
            }

// Get flag character: ? / @
        parseChar   = token[0];
// Check for "?"
        if (parseChar == "?")
            {
            ShowHowToUse("So you want to know?","");
            return FALSE;
            }

// Check for "@" ...
        if (parseChar == "@")        // Read stuff from a file here ...
            {
            token = token.Mid(1);
            CString dataString;
            ReadFileIntoString(token, dataString);  // Need error check ...
            CommandLine = dataString + CommandLine;
            continue;
            }
// Check for "/" ...
        if (parseChar == "/")        // Special processing flag ...
            {
            CString upperToken = token;
            upperToken.MakeUpper();
            if      (upperToken == "/SHOWVALUES")
                ShowRegistrationValues();
            else if (upperToken == "/ASKCONFIGURATION")
                RunConfigurationWizard();
            else if (upperToken == "/NOSPLASH")
                m_bSplashScreen = FALSE;
            else if (upperToken == "/CLEARVALUES")
                m_configVals.RemoveAll();
            else if (upperToken == "/LOADREQUEST")
                {                // For convenience, this same .exe file also is
                SubmitSmartLoadRequest(m_configVals,CommandLine);
                return FALSE;    // the helper-app for a web-browser
                }
            else if (upperToken == "/RESETVALUES")
                m_configVals = GetFactoryDefaultValues();
            else if (upperToken == "/DEBUGBREAK")
                DebugBreak();
            else if (upperToken == "/WRITEVALUES")
                m_configVals.UnparseRegistry(g_Client_Registration_Key);
            else if (upperToken == "/DELETEVALUES")
                DeleteRegistrationValues();
            else if (upperToken == "/QUIT")
                return  FALSE;
// These are debugging/test hacks only ...
            else if (upperToken == "/SIMULATETRANSFER")
                g_bSimulateTransfer = TRUE;
            else if (upperToken == "/SAMPLEDISABLED")
```

```
                g_bSampleDisabled    = TRUE;
            else
                {
                ShowHowToUse
                    ("Invalid command line flag",
                                    token + " " + CommandLine);
                return FALSE;
                }
            continue;
            }
// Something to parse ...
    try {
      m_configVals.ParseLine(token);
    } catch (FormatException ex) {
      ex;
HACKassert("Bad command line value", TRUE);
      MessageBox(NULL, CString("Cannot parse command line argument : ") +
                token, "Command line error", MB_OK);
    }

        }
// Sanity checks on some values ...

    if (!ValidClientPollingPeriod(m_configVals["ClientPollingPeriod"]))
        return InvalidConfigurationValue("ClientPollingPeriod");
    if (!ValidSampleCount(m_configVals["SampleCount"]))
        return InvalidConfigurationValue("SampleCount");
    if (!ValidSampleTime(m_configVals["SampleTime"]))
        return InvalidConfigurationValue("SampleTime");

    if (!ValidSerial(m_configVals["Serial"]))
        return InvalidConfigurationValue("Serial");

    if (!ValidRASPhoneNumber(m_configVals["RASPhoneNumber"]))
        return InvalidConfigurationValue("RASPhoneNumber");
    if (!ValidRASScriptName(m_configVals["RASScriptName"]))
        return InvalidConfigurationValue("RASScriptName");

    if (!ValidBoolean(m_configVals["StartHidden"]))
        return InvalidConfigurationValue("StartHidden");
    if (!ValidBoolean(m_configVals["ConnectionPermanent"]))
        return InvalidConfigurationValue("ConnectionPermanent");
    if (!ValidBoolean(m_configVals["Firewall"]))
        return InvalidConfigurationValue("Firewall");
    if (!ValidBoolean(m_configVals["IPPermanent"]))
        return InvalidConfigurationValue("IPPermanent");
    if (!ValidBoolean(m_configVals["DialAutomatically"]))
        return InvalidConfigurationValue("DialAutomatically");
    return TRUE;
}

AtpAttrib CClient::GetFactoryDefaultValues()
{
    AtpAttrib defaultValues;
    defaultValues.RemoveAll();  // Make sure it is empty.
// Values just for diddling this dialog ...
    defaultValues["StartHidden"]     = "FALSE";

// How we adjust this is not figured out yet ...
    defaultValues["ClientPollingPeriod"]     = "1800";// 30 minutes (seconds)
```

```
// Values set in the CConfigurationDialog ...
    defaultValues["DialAutomatically"]       = "FALSE";
    defaultValues["Firewall"]                = "FALSE";
    defaultValues["IPPermanent"]             = "FALSE";
    defaultValues["ConnectionPermanent"]     = "FALSE";
    defaultValues["TCPProxy"]                = "";
    defaultValues["TCPProxyPortNumber"]      = "80";
    defaultValues["ServerName"]              = DEFAULT_ATP_SERVER;
    defaultValues["RASScriptName"]           = "<undefined>";

// XXX Not needed? Part of RAS script?
    defaultValues["RASPhoneNumber"]          = "7310793";
    defaultValues["RASUserName"]             = "<bad user name>";
    defaultValues["RASPassword"]             = "unfall1";

// Serial number stuff, installation date ...
// Note: "Serial" is what InstallShield uses ...
    defaultValues["Serial"]                  = "0";       // Dummy, not installed

    CTime t = CTime::GetCurrentTime();
    CString debugString;                  // For debugging.
    SetSerialInstallationDate(defaultValues, t, debugString);
    defaultValues["SerialExpirationPeriod"] = "30";      // 30 days.

// Performance threshhold parameters ...
    defaultValues["SampleMinimumTraffic"]    = "300";    // For dial-up.
    defaultValues["SampleThreshholdLittle"]  = "80";
    defaultValues["SampleThreshholdSome"]    = "65";
    defaultValues["SampleThreshholdLots"]    = "50";
    defaultValues["SampleTime"]              = "5000";   // Was 2000 (2 seconds)
    defaultValues["SampleCount"]             = "5";      // Five samples.

// Miscellaneous dialog parameters, etc.
    defaultValues["StatusPeriod"]            = "1500";   // 1.5 seconds.

// Protocol parameters that are not always dynamic ...
    defaultValues["ProtocolBlockSize"]       = "2047";   // Theo's default.

// Added by Theo:
    defaultValues["IDF_RECLAIM_TIME"] = AtpAttrib::UnparseInt(3L*24L*60L*60L);

// This needs to be something other than the temp directory:
// this is just what Theo uses while debugging ...
// The correct value is set up by our installation script.
    char tempDir[2048];          // Read TMP environment value ...
    DWORD status = GetTempPath(sizeof(tempDir)-5, tempDir);
assert (status != 0);
assert (status < sizeof(tempDir)-5);

    CString dirName = CString(tempDir) + "idf"; // What Theo picked ...
    defaultValues["IDF_DIRECTORY"]   = dirName;

// Stuff for SmartLoad requests ...
    dirName = CString(tempDir) + "LoadRequestDir";
    defaultValues["LoadRequestDir"]          = dirName;

    defaultValues["LoadRequestPeriod"]       = "300";    // 5 minutes (seconds)

    dirName = CString(tempDir) + "LoadDeliveryDir";      // default delivery dir.
```

```
    defaultValues["LoadDeliveryDir"]          = dirName;

// Added by Theo:
    AtpAttrib provider, providers;
    provider["ProviderName"] = DEFAULT_ATP_PROVIDER;
    provider["ServerName"] = DEFAULT_ATP_SERVER;
    provider["ServerPassword"] = DEFAULT_ATP_PASSWORD;
    provider["UserName"] = "Hiroshi Kobata";
    provider["UserPassword"] = "another";
    providers[DEFAULT_ATP_PROVIDER] = provider.UnparseLine();

    defaultValues["Providers"] = providers.UnparseLine();

    return defaultValues;
}

VOID CClient::DeleteRegistrationValues()
{
    LONG status;

    status = RegDeleteKey(
        HKEY_LOCAL_MACHINE,                              // open key
        g_Client_Registration_Key);                      // subkey
    if (status != ERROR_SUCCESS)
        {
        SetLastError(status);
HACKassert("Error on RegDeleteKey",status == ERROR_SUCCESS);
        }

    status = RegDeleteKey(
        HKEY_LOCAL_MACHINE,                              // open key
        Registration_Key1);                              // subkey
    if (status != ERROR_SUCCESS)
        {
        SetLastError(status);
HACKassert("Error on RegDeleteKey",status == ERROR_SUCCESS);
        }

    status = RegDeleteKey(
        HKEY_LOCAL_MACHINE,                              // open key
        Registration_Key2);                              // subkey
    if (status != ERROR_SUCCESS)
        {
        SetLastError(status);
HACKassert("Error on RegDeleteKey",status == ERROR_SUCCESS);
        }
}


// Uses a file mapping to see whether we are already running ...
// From Microsoft Knowledge Base article "Allowing Only One
// Application Instance on Win32s" Q123134, Dec '95.

const char * CLIENT_APP_NAME = "ClientAppNameDummyFile";

BOOL AppAlreadyRunning(const char * appName)
{
    HANDLE hMapping  = CreateFileMapping
                      (HANDLE(0xffffffff),     // Special handle for swap-file
                              NULL,            // Security attributes.
```

```
                                PAGE_READONLY,   // File protection
                                0,               // Max size, high-order part.
                                32,              // Max size, low-order  part.
                                appName);        // File name.
    if (hMapping == NULL)
        {
HACKassert("Cannot create file mapping", hMapping != NULL);
        return FALSE;
        }
    if (GetLastError() == ERROR_ALREADY_EXISTS)
        return TRUE;
    return FALSE;
}


VOID CClient::ShowRegistrationValues()
{
    CTextDisplayDlg textDisplayDlg(m_configVals,
                "Current registration values ...");
    textDisplayDlg.DoModal();
}

////////////////////////////////////////////////////////////////////////////
// CClient initialization

BOOL CClient::InitInstance()
{
    // CG: This line was added by the OLE Control Containment component
    AfxEnableControlContainer();

// For debugging / testing use only!
    g_bSimulateTransfer = FALSE;

// Check command line now:
// doing it before "AppAlreadyRunning" is so we can use the command line
// to force registry values to be set, for debugging.

    if (!DoCommandLine())
        return FALSE;

// If an instance of the client is already running, just exit.

    if (AppAlreadyRunning(CLIENT_APP_NAME))
        return FALSE;

// If "special" serial number, serial number still needs to be set ...

    if (m_configVals["Serial"] == "0")
        {
        char message[] =     "Serial number not set yet:\n"
                             " ... code to handle this still needed."
                             ;
        ::MessageBox(NULL, message, "E-Parcel tm Registration Message",
                        MB_OK | MB_ICONEXCLAMATION);
        return FALSE;
        }

// Verify that we have a valid serial number ...
    if (!ValidSerial(m_configVals["Serial"]) ||
                    (m_configVals["Serial"] == "0"))
```

```
        {
        char message[] =    "Your E-Parcel tm client application does not "
                            "appear to be installed properly."
                            "\n\n"
                            "Please be sure that you have installed the "
                            "software correctly, and that you have "
                            "properly entered the serial number "
                            "that was sent to you with the installation "
                            "media."
                            "\n\n"
                            "The E-Parcel tm client application will now exit."
                            ;
        ::MessageBox(NULL, message, "E-Parcel tm Registration Message",
                        MB_OK | MB_ICONEXCLAMATION);

        return FALSE;
        }

// Check expiration date ...

    CTime currentTime = CTime::GetCurrentTime();
    CTime installTime;
    CString textString;
    GetSerialInstallationDate(m_configVals, installTime, textString);
    CTimeSpan elapsedTime = currentTime - installTime;
    int maxDays = AtpAttrib::ParseInt(m_configVals["SerialExpirationPeriod"]);
    if (elapsedTime.GetDays() > maxDays)
        {
        CString message =   "Your E-Parcel tm client application "
                            "appears to be an evaluation copy which has "
                            "expired.\n\n"
                            "This copy was apparently installed on " +
                            textString +
                            "\n\n"
                            "Please contact E-Parcel for a newer version."
                            "\n\n"
                            "The E-Parcel tm client application will now exit."
                            ;
        ::MessageBox(NULL, message, "E-Parcel tm Registration Message",
                        MB_OK | MB_ICONEXCLAMATION);

        return FALSE;
        }

  // Standard initialization
  // If you are not using these features and wish to reduce the size
  //  of your final executable, you should remove from the following
  //  the specific initialization routines you do not need.

// JRW
// Compute a few things from the settings we have, such
// as whether we can do a ClientPoll, etc. ...
// XXX TBD

#ifdef _AFXDLL
  Enable3dControls();           // Call this when using MFC in a shared DLL
#else
  Enable3dControlsStatic(); // Call this when linking to MFC statically
#endif
```

```
#define HAND_CODED_SPLASH  1
#ifdef  HAND_CODED_SPLASH
    if (m_bSplashScreen)
        {
        if (m_splash.Create(IDD_SPLASH))
            {
            m_splash.SetSplashSize();
            m_splash.ShowWindow(SW_SHOW);
            m_splash.UpdateWindow();
            m_splash.SetTimer(1, 500, NULL);
            }
        m_dwSplashTime = ::GetCurrentTime();
        }
#endif  // HAND_CODED_SPLASH

// Put initialization that can take a while here:
// Theo's initialization ...
    if (!AfxSocketInit())
        {
        AfxMessageBox(IDP_SOCKETS_INIT_FAILED);
        return FALSE;
        }

#ifdef  HAND_CODED_SPLASH
    if (m_bSplashScreen)
        {
        while ((::GetCurrentTime() - m_dwSplashTime < 2500) &&
                (m_splash.m_hWnd != NULL))
            Sleep(100);
// timeout expired, destroy the splash window
        if (m_splash.m_hWnd != NULL)
            m_splash.DestroyWindow();
        //m_pMainWnd->UpdateWindow();
        }
#endif  // HAND_CODED_SPLASH

  CClientAppDlg dlg(m_configVals);
  int nResponse = dlg.DoModal();
  if (nResponse == IDOK) {
    // TODO: Place code here to handle when the dialog is
    //  dismissed with OK

  }
  else if (nResponse == IDCANCEL)
    {
// Note: No "cancel" button in our dialog!!!  So how could we get here?
      // TODO: Place code here to handle when the dialog is
      //  dismissed with Cancel
    }


// Clean up our communications stuff ...

    UndialRASConnectionAutomatically();

  // Since the dialog has been closed, return FALSE so that we exit the
  //  application, rather than start the application's message pump.
  return FALSE;
}
```

```
RASCONNSTATE g_DialState(RASCS_Disconnected); // Track progress during dialing

// Callback for RAS connection ...
VOID WINAPI RasDialFunc(UINT unMsg, RASCONNSTATE rasConnState, DWORD dwError)
{

    if (unMsg != WM_RASDIALEVENT)
        {
        CString msg;
        msg.Format("RasDialFunc callback function passed a message that\n"
                   "wasn't WM_RASDIALEVENT: %ld",long(unMsg));
HACKassert(msg,unMsg==WM_RASDIALEVENT);
        }
    if (dwError)
        {
        //only call the function when RAS is installed.
        if(CClient::m_hRasApi32Dll)
            {
            PRasGetErrorString pRasGetErrorString;
            pRasGetErrorString = (PRasGetErrorString)GetProcAddress(CClient::m_hRasApi32Dll
,"RasGetErrorString");
            if(pRasGetErrorString)
                {
                char  szBuf[4096];
                if (pRasGetErrorString((UINT)dwError,(LPSTR)szBuf,sizeof(szBuf)-1) != 0)
                    strcpy(szBuf, "Undefined RAS Callback Error");
                CString msg;
                msg.Format("RAS Callback error: %s: code %ld.",szBuf, (long) dwError);
HACKassert(msg,FALSE);
                }
            }
        }

    g_DialState = rasConnState;
}

// Make a RAS connection ...

// Returns NULL on failure ...

HRASCONN CClient::EstablishRASConnection()
{
    HRASCONN hRasConn;
    PRasDial pRasDial;
    PRasGetErrorString pRasGetErrorString;

    //If RAS is not installed, return immediately.
    if( NULL == m_hRasApi32Dll)
        return NULL;

// Original code taken from \msdev\samples\sdk\win32\rasberry\*.c:
    pRasDial = (PRasDial)GetProcAddress(m_hRasApi32Dll,"RasDial");
    if(NULL == pRasDial)
        return NULL;

    RASDIALPARAMS rdParams;
    DWORD dwRet;
```

```
    // setup RAS Dial Parameters
    rdParams.dwSize = sizeof(RASDIALPARAMS);
    lstrcpy(rdParams.szEntryName, m_configVals["RASScriptName"]);  // Default: first port f
ree
    lstrcpy(rdParams.szPhoneNumber,m_configVals["RASPhoneNumber"]); // Theo's machine: 7310
793
    lstrcpy(rdParams.szCallbackNumber, ""); // No callback, just direct.

    lstrcpy(rdParams.szUserName, m_configVals["RASUserName"]); // ... for now.
    lstrcpy(rdParams.szPassword, m_configVals["RASPassword"]); // ... for now.
    lstrcpy(rdParams.szDomain,   "");        // Default.
    hRasConn = NULL;  // This is required for RasDial to work.

    dwRet = pRasDial(
            NULL,        // No RAS dial extensions
            NULL,        // Default phone book, if we need it.
            &rdParams,   // Dialing paramaters.
            0L,          // RASDIALFUNC notify function being passed  ...
            (RASDIALFUNC) RasDialFunc,  // Notifier callback function.
            &hRasConn);

    if (dwRet)
        {
        pRasGetErrorString = (PRasGetErrorString)GetProcAddress(m_hRasApi32Dll,"RasGetError
String");
        if(pRasGetErrorString)
        {
            char  szBuf[4096];
            if (pRasGetErrorString((UINT)dwRet, (LPSTR)szBuf, sizeof(szBuf)-1) != 0)
                wsprintf((LPSTR)szBuf, "Undefined RAS Dial Error (%ld).", dwRet);

            AfxMessageBox(szBuf);
        }
        hRasConn = NULL;
        }

    return hRasConn;
}

// Hang up current RAS connection

// Returns TRUE on success.

BOOL CClient::DisestablishRASConnection(HRASCONN & hRasConn)
{

// Original code taken from \msdev\samples\sdk\win32\rasberry\*.c:
    DWORD dwRet;
    if(NULL == m_hRasApi32Dll)
        return FALSE;

    PRasHangUp pRasHangUp;
    pRasHangUp = (PRasHangUp)GetProcAddress(m_hRasApi32Dll,"RasHangUp");
    if(NULL == pRasHangUp)
        return FALSE;

    dwRet = pRasHangUp(hRasConn);
    g_DialState = RASCS_Disconnected; // Track progress during dialing

    if (dwRet)
```

```
        {
        PRasGetErrorString pRasGetErrorString;
        pRasGetErrorString = (PRasGetErrorString)GetProcAddress(m_hRasApi32Dll,"RasGetError
String");
        if(pRasGetErrorString)
        {
            char  szBuf[4096];
            if (pRasGetErrorString((UINT)dwRet,(LPSTR)szBuf,sizeof(szBuf)-1) != 0)
                wsprintf((LPSTR)szBuf, "Undefined RAS Dial Error (%ld).", dwRet);

            AfxMessageBox(szBuf);
        }
        hRasConn = NULL;
        return FALSE;
        }

    return TRUE;
}


// Find out what kind of a connection to the network we have,
// so we can do some throughput and performance measurements as
// we go along.

// Note: lpRasConnState used when
// we are first dialing a connection and have some dialing state,
// but no connection yet.
BOOL GetRASConnectionInformation(
        HINSTANCE hRasApi32Dll,          // handle of ras dll
        RASCONNSTATE * lpRasConnState,  // Allowed to be NULL
        DWORD & iRASConnectionCount,
        RASCONNSTATUS & RasConnStatus)
{
    if(FALSE == hRasApi32Dll)
        return FALSE;

    PRasEnumConnections  pRasEnumConnections;

    pRasEnumConnections = (PRasEnumConnections)GetProcAddress(hRasApi32Dll,"RasEnumConnecti
ons");
    if(NULL == pRasEnumConnections)
        return FALSE;

    RasConnStatus.dwSize = sizeof(RASCONNSTATUS);
    RASCONN RASConnection[1];
    RASConnection[0].dwSize = sizeof(RASCONN);
    DWORD rasBufSize         = sizeof(RASCONN);

    DWORD status = pRasEnumConnections(
            &RASConnection[0],       // buffer to receive connections data
            &rasBufSize,             // size in bytes of buffer
            &iRASConnectionCount);  // number of connections written to buffer
    if (status != 0)
        {
        CString msg;
        msg.Format("Status error on RasEnumConnections: %ld", long(status));
HACKassert(msg,status == 0);
        }
    if (iRASConnectionCount > 1)
```

```
        {
        CString msg;
        msg.Format("More than one RAS connection: not handled yet: %ld",
                   long(iRASConnectionCount));
HACKassert(msg,iRASConnectionCount <= 1);    // Don't handle the case of multiple
                                             // RAS connections yet ...

        }
    if (iRASConnectionCount == 0)
        {
// Nothing connected: if we are dialing, we have passed in an lpRasConnState:
        if (lpRasConnState == NULL)
            RasConnStatus.rasconnstate = RASCS_Disconnected;     // Not dialing
        else
            RasConnStatus.rasconnstate = *lpRasConnState;        // Still dialing
        }
    else{
        PRasGetConnectStatus pRasGetConnectStatus;

        pRasGetConnectStatus = (PRasGetConnectStatus)GetProcAddress(hRasApi32Dll,"RasGetCon
nectStatus");
        if(NULL == pRasGetConnectStatus)
            return FALSE;

        status = pRasGetConnectStatus(RASConnection[0].hrasconn,
                                      &RasConnStatus);
        if (status != 0)
            {
            CString msg;
            msg.Format("Invalid status on RasGetConnectionStatus:  %ld",
                       long(status));
HACKassert(msg,status == 0);
            }
        }

    return TRUE;
}

// Used by "dial automatically" on polling:
// If we are supposed to dial in, and the user said to dial
// automatically, then go dial!
// NOTE: We don't know when to hang up automatically: some other program
// may be using the RAS connection in the meantime (like PointCast).
// The best solution might be for the user to give us a RAS connection name
// that has a time-out hang-up after 15 minutes or something.

void CClient::DialRASConnectionAutomatically()
{

    if (AtpAttrib::ParseBool(m_configVals["ConnectionPermanent"]))
        return;      // Permanent connection, don't use RAS dial-up.
    if (AtpAttrib::ParseBool(m_configVals["DialAutomatically"]) != TRUE)
        return;      // Not supposed to auto-dial.

    m_hRasConn  = NULL;

// Do we already have a RAS connection?
    m_bRASAlreadyActive = FALSE;
    DWORD iRasConCount; // == 0 if no RAS connection running.
    RASCONNSTATUS RasConStat;
    if (!GetRASConnectionInformation(m_hRasApi32Dll,NULL,iRasConCount,RasConStat))
```

```
        {
HACKassert("XXX Cannot get connection information",FALSE);
        }
    m_bRASAlreadyActive = (iRasConCount > 0);

// O.K., dial it,

    if (m_bRASAlreadyActive)
        return;      // Already have a RAS connection running.

    m_hRasConn = EstablishRASConnection();
    if (NULL == m_hRasConn)
        {
        AfxMessageBox("XXX Cannot dial RAS");
        }
}

// Note: no way to check for the connection being used by some
// other application -- perhaps we should just have the user give us a
// RAS connection with an automatic hang-up timeout, and leave it at that.

void CClient::UndialRASConnectionAutomatically()
{
    if (AtpAttrib::ParseBool(m_configVals["ConnectionPermanent"]))
        return;      // Permanent connection, don't use RAS dial-up.
    if (AtpAttrib::ParseBool(m_configVals["DialAutomatically"]) != TRUE)
        return;      // Not supposed to auto-dial.
    if (m_bRASAlreadyActive)
        return;      // Already had a RAS connection running,
                     // so this wasn't auto-dialed..

    if (!DisestablishRASConnection(m_hRasConn))
        {
        AfxMessageBox("XXX Cannot hang up RAS");
        }
}

CClient::~CClient()    // Destructor.
    {
    FreeLibrary(m_hRasApi32Dll);
    m_hRasApi32Dll = NULL;
    }

BOOL CClient::PreTranslateMessage(MSG* pMsg)
{
    BOOL bResult = CWinApp::PreTranslateMessage(pMsg);
#ifdef  HAND_CODED_SPLASH
    if (m_bSplashScreen)
        {
        if (m_splash.m_hWnd != NULL &&
            (pMsg->message == WM_KEYDOWN ||
             pMsg->message == WM_SYSKEYDOWN ||
             pMsg->message == WM_LBUTTONDOWN ||
             pMsg->message == WM_RBUTTONDOWN ||
             pMsg->message == WM_MBUTTONDOWN ||
             pMsg->message == WM_NCLBUTTONDOWN ||
             pMsg->message == WM_NCRBUTTONDOWN ||
             pMsg->message == WM_NCMBUTTONDOWN))
            {
            m_splash.DestroyWindow();
```

```
            //m_pMainWnd->UpdateWindow();
            }
        }
#endif  // HAND_CODED_SPLASH

    return bResult;
}

BOOL CClient::OnIdle(LONG lCount)
{
    // call base class idle first
    BOOL bResult = CWinApp::OnIdle(lCount);

#ifdef  HAND_CODED_SPLASH
    if (m_bSplashScreen)
        {   // then do our work
        if (m_splash.m_hWnd != NULL)
            {
            if (::GetCurrentTime() - m_dwSplashTime > 2500 + 100)
                {
                // timeout expired, destroy the splash window
                m_splash.DestroyWindow();
                //m_pMainWnd->UpdateWindow();

            // NOTE: don't set bResult to FALSE,
            //  CWinApp::OnIdle may have returned TRUE
            }
            else
                bResult = TRUE; // check again later...
            }
        }
#endif  // HAND_CODED_SPLASH

    return bResult;
}


// Routine to do our "set configuration" Wizard dialog stuff ...

void RunConfigurationWizard()
{
    CPropertySheet dlgPropertySheet("XXX Put Title Here");

// Temp copy of current settings ...
    AtpAttrib tconfigVals = theApp.m_configVals;

// Wizard pages:
// Note: I wanted to pass tconfigVals in the contructors, but
// that was causing problems with the IMPLEMENT_DYNCREATE macro for
// CPropertyPage.

    CPropertyPageInternetConnection pageInternetConnection;
    dlgPropertySheet.AddPage(&pageInternetConnection);
    pageInternetConnection.SetConfigVals(&tconfigVals);

    CPropertyPageIPAssignment pageIPAssignment;
    dlgPropertySheet.AddPage(&pageIPAssignment);
    pageIPAssignment.SetConfigVals(&tconfigVals);

    CPropertyPageDialAutomatically pageDialAutomatically;
```

```
    dlgPropertySheet.AddPage(&pageDialAutomatically);
    pageDialAutomatically.SetConfigVals(&tconfigVals);

    CPropertyPageFirewall pageFirewall;
    dlgPropertySheet.AddPage(&pageFirewall);
    pageFirewall.SetConfigVals(&tconfigVals);

    CPropertyPageProxyInfo pageProxyInfo;
    dlgPropertySheet.AddPage(&pageProxyInfo);
    pageProxyInfo.SetConfigVals(&tconfigVals);

    CPropertyPageRasScript pageRasScript;
    dlgPropertySheet.AddPage(&pageRasScript);
    pageRasScript.SetConfigVals(&tconfigVals);

    dlgPropertySheet.SetWizardMode();

    int iResult = dlgPropertySheet.DoModal();
HACKassert("Bad return from property sheet",
                    (iResult==IDCANCEL) || (iResult == ID_WIZFINISH));

    if (iResult == ID_WIZFINISH)     // Set the new values ...
        {
        theApp.m_configVals.Merge(tconfigVals);
        theApp.m_configVals.UnparseRegistry(g_Client_Registration_Key);
        }
}

// This routine derives from stuff from the configuration results,
// such as whether we can/should have the client initiate connections (rather
// than just the server doing this), whether we want to auto-dial, etc.

void ComputeConfigurationResults()
{

}


#ifdef  SOURCE_CONTROL_BLOCK
 Project Name: E-PS Client/Server program
 E-PS Inc.
 Copyright 1996, 1997. All Rights Reserved.
 SUBSYSTEM: ClientApp customer client program
 $Workfile: ClientApp.cpp $
 $Author: Tding $
 $Revision: 56 $
 $Date: 2/03/97 10:26a $
 $Modtime: 2/03/97 10:26a $

 $History: ClientApp.cpp $
 *
 * *****************  Version 56  *****************
 * User: Tding         Date: 2/03/97    Time: 10:26a
 * Updated in $/atp/ClientApp
 * Changes made for no RAS situation.
 *
 * *****************  Version 55  *****************
 * User: Jrward        Date: 1/31/97    Time: 2:48p
 * Updated in $/atp/ClientApp
 * Global check-in.
```

```
*
* ****************  Version 54  ****************
* User: Jrward        Date: 1/31/97    Time: 10:47a
* Updated in $/atp/ClientApp
* Stubbed out some netstat asserts for now.
*
* ****************  Version 53  ****************
* User: Jrward        Date: 1/30/97    Time: 1:48p
* Updated in $/atp/ClientApp
* Added /SampleDisabled switch, other stuff for convenience, so Theo can
* test his protocol code on his own machine.
*
* ****************  Version 52  ****************
* User: Jrward        Date: 1/29/97    Time: 5:30p
* Updated in $/atp/ClientApp
* End-Of-Day checkin festival.
*
* ****************  Version 51  ****************
* User: Jrward        Date: 1/28/97    Time: 1:11p
* Updated in $/atp/ClientApp
* Afternoon check-in -- mostly fixes to installation from all the
* SmartLoad fixes.
*
* ****************  Version 49  ****************
* User: Jrward        Date: 1/27/97    Time: 10:29a
* Updated in $/atp/ClientApp
* Smartload stuff now in separate sources SmartLoad.cpp, SmartLoad.h
*
* ****************  Version 48  ****************
* User: Jrward        Date: 1/24/97    Time: 4:58p
* Updated in $/atp/ClientApp
* New .epsclient SmartLoad file handling: Just waiting on Theo's stuff,
* which he hasn't tested yet.
*
* ****************  Version 47  ****************
* User: Jrward        Date: 1/23/97    Time: 4:50p
* Updated in $/atp/ClientApp
* Daily check-in: looks like I have the SmartLoad stuff working, I just
* need the new Client-side protocol code from Theo.
*
* ****************  Version 46  ****************
* User: Jrward        Date: 1/23/97    Time: 3:04p
* Updated in $/atp/ClientApp
* SmartLoad request files handled at the file level (*.epsclient in
* LoadRequestDir directory), but no processing of the innards of the
* files yet.
*
* ****************  Version 45  ****************
* User: Jrward        Date: 1/23/97    Time: 12:25p
* Updated in $/atp/ClientApp
* Added /LoadRequest switch on clientapp.exe, so can use it as "helper"
* app with Web browsers for SmartLoad.  Actual SmartLoad processing not
* done yet.
*
* ****************  Version 44  ****************
* User: Jrward        Date: 1/20/97    Time: 12:37p
* Updated in $/atp/ClientApp
* Change registry version key from 961130 to 970120, to reflect changes
* in Theo's protocol code.
*
```

```
 * *****************  Version 43  *****************
 * User: Jrward        Date: 1/15/97    Time: 5:14p
 * Updated in $/atp/ClientApp
 * End of the day check-in festival.
 *
 * *****************  Version 42  *****************
 * User: Jrward        Date: 1/13/97    Time: 5:02p
 * Updated in $/atp/ClientApp
 * End-of-the-day checkin.
 *
 * *****************  Version 41  *****************
 * User: Jrward        Date: 1/13/97    Time: 5:00p
 * Updated in $/atp/ClientApp
 * Cannot get ODBC calls in fake JRDemo.exe to work from the
 * ServerISAPI.DLL: work fine when running www server from debugger, but
 * throw when opening the CustomerDB.mdb database when running normally as
 * a service!
 *
 * *****************  Version 40  *****************
 * User: Jrward        Date: 1/13/97    Time: 9:36a
 * Updated in $/atp/ClientApp
 * Small change in "Setup" dialog for setting IDF directory: still more to
 * do.
 *
 * *****************  Version 37  *****************
 * User: Ttonchev      Date: 1/07/97    Time: 4:27a
 * Updated in $/atp/ClientApp
 *
 * *****************  Version 36  *****************
 * User: Jrward        Date: 1/07/97    Time: 12:00a
 * Updated in $/atp/ClientApp
 * StatusPeriod now configurable.
 *
 * *****************  Version 35  *****************
 * User: Jrward        Date: 1/06/97    Time: 2:03p
 * Updated in $/atp/ClientApp
 * Change default server to "theserver"
 *
 * *****************  Version 34  *****************
 * User: Jrward        Date: 1/06/97    Time: 1:21p
 * Updated in $/atp/ClientApp
 * Splash screen can now be suppressed.
 *
 * *****************  Version 33  *****************
 * User: Ttonchev      Date: 1/04/97    Time: 4:07p
 * Updated in $/atp/ClientApp
 * Added some more support for multiple providers
 *
 * *****************  Version 32  *****************
 * User: Jrward        Date: 1/04/97    Time: 2:48p
 * Updated in $/atp/ClientApp
 * Small changes for startup splash screen: code not done, so #ifdef'ed
 * out in certain places for now.
 *
 * *****************  Version 30  *****************
 * User: Jrward        Date: 1/02/97    Time: 10:57a
 * Updated in $/atp/ClientApp
 * Added SOURCE_CONTROL_BLOCK stuff.

// JRW 961128   Change command-line, registry stuff to use
```

```
//                extended AtpAttrib class.
//                Changed version from 961117 to 961130.
// JRW 961203     Small debugging changed.
// JRW 961204     SampleThreshhold* values now in configuration data,
//                mostly so we can dickie with them while testing.
// JRW 961212     Put RunConfigurationWizard here, so it can be invoked from
//                command line as part of our installation procedures.
// JRW 961215     Moved "connect to server at startup" to pollling loop in
//                ClientAppDlg: it was too hard getting the non-Window
//                SetTimer functions to work with the application class.
// JRW 961220     Moved protocol initialization, shut-down from ClientApp.cpp
//                to ClientAppDlg.cpp, to fix the problem of the update
//                call-back function being called after the main dialog was
//                already destroyed.
#endif     // SOURCE_CONTROL_BLOCK
```

```
// ClientApp.h : main header file for the CLIENTAPP application

#ifndef CLIENTAPP_H_INCLUDED
#define CLIENTAPP_H_INCLUDED


#ifndef __AFXWIN_H__
#error include 'stdafx.h' before including this file for PCH
#endif
#include <ras.h>     // JRW

#include "resource.h"        // main symbols
#include "SplashDlg.h"
#include "callback.h"
#include "options.h"
#include "idfserver.h"
#include "atpsock.h"
#include "socketcontrol.h"
#include "atpattrib.h"  // For command-line options, etc.
#include "ContentControl.h" // TGT 961130
#include <afxdisp.h> // CG: added by OLE Control Containment component

// So that different modules will report the same build date ...
extern CString g_ClientAppBuildDate;

typedef DWORD (FAR PASCAL * PRasGetErrorString)(
    UINT uErrorValue,    // error to get string for
    LPTSTR lpszErrorString,      // buffer to hold error string
    DWORD cBufSize       // size, in characters, of buffer
   );

typedef DWORD (FAR PASCAL * PRasDial)(
    LPRASDIALEXTENSIONS lpRasDialExtensions,     // pointer to function extensions data
    LPTSTR lpszPhonebook,        // pointer to full path and filename of phonebook file
    LPRASDIALPARAMS lpRasDialParams,     // pointer to calling parameters data
    DWORD dwNotifierType,        // specifies type of RasDial event handler
    LPVOID lpvNotifier, // specifies a handler for RasDial events
    LPHRASCONN lphRasConn        // pointer to variable to receive connection handle
   );

typedef DWORD (FAR PASCAL * PRasHangUp)(
    HRASCONN hrasconn    // handle to the RAS connection to hang up
   );

typedef DWORD (FAR PASCAL * PRasGetConnectStatus)(
    HRASCONN hrasconn,  // handle to RAS connection of interest
    LPRASCONNSTATUS lprasconnstatus      // buffer to receive status data
   );

typedef DWORD (FAR PASCAL * PRasEnumConnections)(
    LPRASCONN lprasconn,         // buffer to receive connections data
    LPDWORD lpcb,        // size in bytes of buffer
    LPDWORD lpcConnections       // number of connections written to buffer
  );

typedef DWORD (FAR PASCAL * PRasEnumEntries)(
    LPTSTR reserved,         // reserved, must be NULL
    LPTSTR lpszPhonebook,    // pointer to full path and filename of phonebook file
    LPRASENTRYNAME lprasentryname,  // buffer to receive phonebook entries
    LPDWORD lpcb,            // size in bytes of buffer
```

```
    LPDWORD lpcEntries       // number of entries written to buffer
   );

////////////////////////////////////////////////////////////////////////////
// CClient:
// See Client.cpp for the implementation of this class
//

class CClient : public CWinApp
{
public:
  CClient();
  ~CClient();  // Destructor.

  // Overrides
  // ClassWizard generated virtual function overrides
  //{{AFX_VIRTUAL(CClient)
    public:
    virtual BOOL InitInstance();
    virtual BOOL OnIdle(LONG lCount);
    virtual BOOL PreTranslateMessage(MSG* pMsg);
    //}}AFX_VIRTUAL

  // Implementation

private:
    VOID ShowHowToUse(const char * msg, const char * BadCommandLinePart);
                                  // Show usage in a message box
    BOOL DoCommandLine();         // Parse out command line.
    VOID ShowRegistrationValues();
    AtpAttrib GetFactoryDefaultValues();
    VOID DeleteRegistrationValues();

    BOOL ValidBoolean(CString & sBoolean);
    BOOL ValidRASScriptName(CString & RASScriptName);
    BOOL ValidSerial(CString & serialNumber);
    BOOL ValidClientPollingPeriod(CString & sampleTime);
    BOOL ValidSampleTime(CString & sampleTime);
    BOOL ValidSampleCount(CString & sampleCount);
    BOOL ValidRASPhoneNumber(CString & phoneNumber);

// For splash screen ...

    DWORD m_dwSplashTime;
    CSplashDlg m_splash;

public:
// Command line / registry configuration parameters ...
    AtpAttrib   m_configVals;
    BOOL m_bSplashScreen;   // Used to suppress splash screen.

// JRW These should be in their own class?
//      ... or at least not part of theApp?
    void   DialRASConnectionAutomatically();
    void UndialRASConnectionAutomatically();
    BOOL DisestablishRASConnection(HRASCONN & hRasConn);
    HRASCONN EstablishRASConnection();
    BOOL    m_bRASAlreadyActive;
    HRASCONN m_hRasConn;     // Ras connection we are using, if any.
```

```
//  RAS DLL
    static HINSTANCE m_hRasApi32Dll;

private:
    BOOL InvalidConfigurationValue(const char * name);
    VOID SetSerialInstallationDate(AtpAttrib & configVals, CTime & t,
                    CString & textString);
    VOID GetSerialInstallationDate(AtpAttrib & configVals, CTime & t,
                    CString & textString);


  //{{AFX_MSG(CClient)
  // NOTE - the ClassWizard will add and remove member functions here.
  //    DO NOT EDIT what you see in these blocks of generated code !
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP();
};


extern BOOL GetRASConnectionInformation(
             HINSTANCE hRasApi32Dll,           // handle of ras dll
             RASCONNSTATE * lpRasConnState,    // State passed in while dialing
             DWORD & iRASConnectionCount,      // Returns count of connecitons
             RASCONNSTATUS & RasConnStatus); // Returns status

extern void RunConfigurationWizard();

extern void ComputeConfigurationResults();

extern CClient theApp;  // JRW: anyplace we use this, we should probably
                        // look into making the code more modular!
extern const char * g_Client_Registration_Key;

// For debugging and testing only ...
extern BOOL g_bSimulateTransfer;     // Simulate transfer of data for dialog
extern BOOL g_bSampleDisabled;       // Disable "network sample" measurements
                                     // for our "politeness" on busy networks.


#ifdef  SOURCE_CONTROL_BLOCK
 Project Name: E-PS Client/Server program
 E-PS Inc.
 Copyright 1996, 1997. All Rights Reserved.
 SUBSYSTEM: ClientApp customer client program
 $Workfile: ClientApp.h $
 $Author: Tding $
 $Revision: 37 $
 $Date: 2/03/97 10:27a $
 $Modtime: 2/02/97 4:30p $

 $History: ClientApp.h $
 *
 * *****************  Version 37  *****************
 * User: Tding         Date: 2/03/97    Time: 10:27a
 * Updated in $/atp/ClientApp
 * Added m_hRasApi32Dll as a static handle to the RAS library.
 *
 * *****************  Version 36  *****************
 * User: Jrward        Date: 1/31/97    Time: 2:48p
 * Updated in $/atp/ClientApp
```

```
* Global check-in.
*
* *****************  Version 35  *****************
* User: Jrward        Date: 1/31/97     Time: 10:47a
* Updated in $/atp/ClientApp
* Stubbed out some netstat asserts for now.
*
* *****************  Version 34  *****************
* User: Jrward        Date: 1/30/97     Time: 1:48p
* Updated in $/atp/ClientApp
* Check everything I have in, so Theo can get it.
*
* *****************  Version 33  *****************
* User: Jrward        Date: 1/29/97     Time: 5:30p
* Updated in $/atp/ClientApp
* End-Of-Day checkin festival.
*
* *****************  Version 32  *****************
* User: Jrward        Date: 1/28/97     Time: 1:11p
* Updated in $/atp/ClientApp
* Afternoon check-in -- mostly fixes to installation from all the
* SmartLoad fixes.
*
* *****************  Version 31  *****************
* User: Jrward        Date: 1/27/97     Time: 5:11p
* Updated in $/atp/ClientApp
* SmartLoad sort of working ...
*
* *****************  Version 30  *****************
* User: Jrward        Date: 1/27/97     Time: 10:29a
* Updated in $/atp/ClientApp
* Smartload stuff now in separate sources SmartLoad.cpp, SmartLoad.h
*
* *****************  Version 29  *****************
* User: Jrward        Date: 1/24/97     Time: 4:58p
* Updated in $/atp/ClientApp
* New .epsclient SmartLoad file handling: Just waiting on Theo's stuff,
* which he hasn't tested yet.
*
* *****************  Version 28  *****************
* User: Jrward        Date: 1/23/97     Time: 4:50p
* Updated in $/atp/ClientApp
* Daily check-in: looks like I have the SmartLoad stuff working, I just
* need the new Client-side protocol code from Theo.
*
* *****************  Version 27  *****************
* User: Jrward        Date: 1/23/97     Time: 3:04p
* Updated in $/atp/ClientApp
* SmartLoad request files handled at the file level (*.epsclient in
* LoadRequestDir directory), but no processing of the innards of the
* files yet.
*
* *****************  Version 26  *****************
* User: Jrward        Date: 1/23/97     Time: 12:25p
* Updated in $/atp/ClientApp
* Added /LoadRequest switch on clientapp.exe, so can use it as "helper"
* app with Web browsers for SmartLoad.  Actual SmartLoad processing not
* done yet.
*
* *****************  Version 25  *****************
```

```
 * User: Jrward        Date: 1/15/97     Time: 5:15p
 * Updated in $/atp/ClientApp
 * End of the day check-in festival.
 *
 * *****************  Version 24  *****************
 * User: Jrward        Date: 1/13/97     Time: 5:02p
 * Updated in $/atp/ClientApp
 * End-of-the-day checkin.
 *
 * *****************  Version 23  *****************
 * User: Jrward        Date: 1/13/97     Time: 5:00p
 * Updated in $/atp/ClientApp
 * Cannot get ODBC calls in fake JRDemo.exe to work from the
 * ServerISAPI.DLL: work fine when running www server from debugger, but
 * throw when opening the CustomerDB.mdb database when running normally as
 * a service!
 *
 * *****************  Version 22  *****************
 * User: Jrward        Date: 1/13/97     Time: 9:36a
 * Updated in $/atp/ClientApp
 * Small change in "Setup" dialog for setting IDF directory: still more to
 * do.
 *
 * *****************  Version 21  *****************
 * User: Jrward        Date: 1/10/97     Time: 9:42a
 * Updated in $/atp/ClientApp
 * More stuff hooked up in Setup dialog, but still need to work on IDF
 * directory selection.
 *
 * *****************  Version 20  *****************
 * User: Jrward        Date: 1/06/97     Time: 1:21p
 * Updated in $/atp/ClientApp
 * Splash screen can now be suppressed.
 *
 * *****************  Version 17  *****************
 * User: Jrward        Date: 1/02/97     Time: 10:57a
 * Updated in $/atp/ClientApp
 * Added SOURCE_CONTROL_BLOCK stuff.
 *
 * *****************  Version 16  *****************
 * User: Jrward        Date: 1/02/97     Time: 10:26a
 * Updated in $/atp/ClientApp
 *
 * *****************  Version 15  *****************
 * User: Jrward        Date: 1/02/97     Time: 10:25a
 * Updated in $/atp/ClientApp
 *
 * *****************  Version 13  *****************
 * User: Jrward        Date: 1/02/97     Time: 10:16a
 * Updated in $/atp/ClientApp
 * Test of SOURCE_CONTROL_BLOCK
// JRW 961128   Changed to use AtpAttrib to hold the configuration values
//              from the command line / registry.
//              HACKassert moved to utility.h
// JRW 961212   Put RunConfigurationWizard here, so it can be invoked from
//              command line as part of our installation procedures.
// JRW 961215   Moved "connect to server at startup" to pollling loop in
//              ClientAppDlg: it was too hard getting the non-Window
//              SetTimer functions to work with the application class.
// JRW 961220   Moved protocol initialization, shut-down from ClientApp.cpp
```

```
//              to ClientAppDlg.cpp, to fix the problem of the update
//              call-back function being called after the main dialog was
//              already destroyed:
//                  m_idfserver, m_atpserver, m_contentserver
#endif     // SOURCE_CONTROL_BLOCK

#endif   // CLIENTAPP_H_INCLUDED
```

```
// ClientAppDlg.cpp : implementation file

#include "stdafx.h"
#include <assert.h> // JRW for debugging.

#include "utility.h"
#include "atpattrib.h"
#include "atpexcept.h"

#include "ClientApp.h"
#include "ClientAppDlg.h"
#include "SmartLoad.h"
#include "atpsock.h"    // for AtpState.

#include "ProcessMonitor.h" // for the UGLY stuff ...

#include "ConfigureDlg.h"
#include "DbgConfigDlg.h"
#include "EasterEggDlg.h"
#include "TextDisplayDlg.h"

extern RASCONNSTATE g_DialState;    // JRW Ungly hack.


#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
  CAboutDlg();

  // Dialog Data
  //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    CStatic m_ICON_IMAGE;
    //}}AFX_DATA

  // ClassWizard generated virtual function overrides
  //{{AFX_VIRTUAL(CAboutDlg)
protected:
  virtual VOID DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
  //}}AFX_VIRTUAL

  // Implementation
protected:
  //{{AFX_MSG(CAboutDlg)
    afx_msg VOID OnRButtonDblClk(UINT nFlags, CPoint point);
    //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
    };

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
```

```
  //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

VOID CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
  CDialog::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CAboutDlg)
    DDX_Control(pDX, IDC_ICON_IMAGE, m_ICON_IMAGE);
    //}}AFX_DATA_MAP
}

// Check for a double-right-click on the little "company icon" image in
// the upper-left of the dialog.
VOID CAboutDlg::OnRButtonDblClk(UINT nFlags, CPoint point)
{
// Get rect for our secret icon, and convert to client co-ordinates for the
// main dialog, to match the mouse point location:

    CRect iconRect;
    m_ICON_IMAGE.GetWindowRect(&iconRect);  // Returns screen coordinates
    ScreenToClient(&iconRect);       // Convert to dialog's client co-ordinates

// Now that the mouse point and icon rect are in the same co-ordinate system,
// check whether the user clicked in the icon:

    BOOL bInside = iconRect.PtInRect(point);

// If they clicked inside, and they are holding down both the
// control and the shift keys,  bring up our secret feature ...

    if (bInside && (nFlags & MK_CONTROL) && (nFlags & MK_SHIFT))
        {
        CEasterEggDlg dlgEasterEgg;
        dlgEasterEgg.DoModal();
        }

// "Now we return to your previously scheduled programming, in progress ..."

    CDialog::OnRButtonDblClk(nFlags, point);
}


BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
  //{{AFX_MSG_MAP(CAboutDlg)
    ON_WM_RBUTTONDBLCLK()
    //}}AFX_MSG_MAP
  END_MESSAGE_MAP();

////////////////////////////////////////////////////////////////////////////
// CClientAppDlg dialog

#define MYWM_NOTIFYICON (WM_USER+100)

// MyTaskBarAddIcon - adds an icon to the taskbar notification area.
// Returns TRUE if successful or FALSE otherwise.
// hwnd - handle of the window to receive callback messages
// uID - identifier of the icon
// hicon - handle of the icon to add
// lpszTip - tooltip text
```

```
BOOL MyTaskBarAddIcon(HWND hwnd, UINT uID, HICON hicon, LPSTR lpszTip)
{
    BOOL res;
    NOTIFYICONDATA tnid;

    tnid.cbSize = sizeof(NOTIFYICONDATA);
    tnid.hWnd = hwnd;
    tnid.uID = uID;
    tnid.uFlags = NIF_MESSAGE | NIF_ICON | NIF_TIP;
    tnid.uCallbackMessage = MYWM_NOTIFYICON;
    tnid.hIcon = hicon;
    if (lpszTip)
        lstrcpyn(tnid.szTip, lpszTip, sizeof(tnid.szTip));
    else
        tnid.szTip[0] = '\0';

    res = Shell_NotifyIcon(NIM_ADD, &tnid);

    if (hicon)
        DestroyIcon(hicon);

    return res;
}

// MyTaskBarDeleteIcon - deletes an icon from the taskbar
//     notification area.
// Returns TRUE if successful or FALSE otherwise.
// hwnd - handle of the window that added the icon
// uID - identifier of the icon to delete
BOOL MyTaskBarDeleteIcon(HWND hwnd, UINT uID)
{
    BOOL res;
    NOTIFYICONDATA tnid;

    tnid.cbSize = sizeof(NOTIFYICONDATA);
    tnid.hWnd = hwnd;
    tnid.uID = uID;

    res = Shell_NotifyIcon(NIM_DELETE, &tnid);
    return res;
}


// Design note:
// Some of these loops could be moved to independent threads,
// but we haven't done so right now, because
// 1) We might have to make them thread-safe.
// 2) We might have to make some of the member-data objects
//    they reference thread-safe.
// 3) These loops repeat (at most) every few minutes, which doesn't
//    sound much like the stuff that time-critical threads are made of.

// Event ID's for OnTimer function ...

#define EVENT_PROGRESS              101    // progress-bar update
#define EVENT_POLL_SERVER           102    // Poll the server ...
#define EVENT_LOAD_REQUEST          103    // Check LoadRequest directory

#define INTERVAL_INITIAL          (2000L)  // Initial timer periods are short,
                                           // so that first updates are quick,
```

```
                                                    // but after the dialog is initialized
VOID CClientAppDlg::InitializeLoopTimers()
{

// Set up our repeating functions that run off of timers ...

// Progress bar:
// JRW Set up a timer, just to catch (eventually) any changes in state
// the call-back functions may skip due to quirks in the evolving transfer
// code ...

// Note: first timer period is short ...
    m_IDTimerProgress = SetTimer(EVENT_PROGRESS, INTERVAL_INITIAL, NULL);
HACKassert("Could not create progress bar timer",m_IDTimerProgress != 0);

// Set up our polling loop for our periodic client-initiated connections
// with the server ...

    int pollTime = AtpAttrib::ParseInt(m_configVals["ClientPollingPeriod"]);
    if (pollTime > 0)
        {
// Note: first timer period is short ...
        m_IDTimerPollServer = SetTimer(EVENT_POLL_SERVER,INTERVAL_INITIAL,NULL);
HACKassert("Could not create polling timer",m_IDTimerPollServer != 0);
        }
    else
      LoopClientPollServer(); // Check in with the designated server ... (TGT)

    int reqTime = AtpAttrib::ParseInt(m_configVals["LoadRequestPeriod"]);
    if (reqTime > 0)
        {
        m_IDTimerLoadRequest=SetTimer(EVENT_LOAD_REQUEST,INTERVAL_INITIAL,NULL);
HACKassert("Could not create request timer",m_IDTimerLoadRequest != 0);
        }

}

VOID CClientAppDlg::KillLoopTimers()
{
    if (!KillTimer(m_IDTimerProgress))
        {
HACKassert("Could not kill progress bar timer",FALSE);
        }
    if (!KillTimer(m_IDTimerPollServer))
        {
HACKassert("Cannot kill polling timer",FALSE);
        }
    if (!KillTimer(m_IDTimerLoadRequest))
        {
HACKassert("Cannot kill load request timer",FALSE);
        }
}

CClientAppDlg::CClientAppDlg(AtpAttrib & configVals, CWnd* pParent /*=NULL*/)
  : CDialog(CClientAppDlg::IDD, pParent),
            m_configVals(configVals),
            m_bStartHidden(AtpAttrib::ParseBool(m_configVals["StartHidden"])),
            m_bSecretDebugMode(FALSE)
{
  //{{AFX_DATA_INIT(CClientAppDlg)
```

```
    //}}AFX_DATA_INIT
  // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
  m_hIcon16 = AfxGetApp()->LoadIcon(IDI_DOCICON32);
  m_hIcon32 = AfxGetApp()->LoadIcon(IDI_DOCICON32);

}

VOID CClientAppDlg::DoDataExchange(CDataExchange* pDX)
{
  CDialog::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CClientAppDlg)
    DDX_Control(pDX, IDC_PROGRESS_LABEL10, m_ProgressLabel10);
    DDX_Control(pDX, IDC_SECRET_DEBUG_TEXT_TOP, m_SECRET_DEBUG_TEXT_TOP);
    DDX_Control(pDX, IDC_SECRET_DEBUG_TEXT_BOTTOM, m_SECRET_DEBUG_TEXT_BOTTOM);
    DDX_Control(pDX, IDC_ICON_IMAGE, m_ICON_IMAGE);
    DDX_Control(pDX, IDC_PROGRESS_LABEL9, m_ProgressLabel9);
    DDX_Control(pDX, IDC_PROGRESS_LABEL8, m_ProgressLabel8);
    DDX_Control(pDX, IDC_PROGRESS_LABEL7, m_ProgressLabel7);
    DDX_Control(pDX, IDC_PROGRESS_LABEL6, m_ProgressLabel6);
    DDX_Control(pDX, IDC_PROGRESS_LABEL5, m_ProgressLabel5);
    DDX_Control(pDX, IDC_PROGRESS_LABEL4, m_ProgressLabel4);
    DDX_Control(pDX, IDC_PROGRESS_LABEL3, m_ProgressLabel3);
    DDX_Control(pDX, IDC_PROGRESS_LABEL2, m_ProgressLabel2);
    DDX_Control(pDX, IDC_PROGRESS_LABEL1, m_ProgressLabel1);
    DDX_Control(pDX, IDC_PROGRESS_BAR,    m_PROGRESS_BAR);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CClientAppDlg, CDialog)
  //{{AFX_MSG_MAP(CClientAppDlg)
  ON_WM_SYSCOMMAND()
  ON_WM_DESTROY()
  ON_WM_PAINT()
  ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_HIDE, OnHide)
    ON_WM_TIMER()
    ON_WM_CLOSE()
    ON_BN_CLICKED(ID_CONFIGURE, OnConfigure)
    ON_WM_RBUTTONDBLCLK()
    ON_BN_CLICKED(IDC_BUTTON_ABOUT, OnButtonAbout)
    ON_BN_CLICKED(IDC_BUTTON_HELP, OnButtonHelp)
    ON_BN_CLICKED(IDC_BUTTON_CLIENT_POLL_NOW, OnButtonClientPollServerNow)
    ON_BN_CLICKED(ID_DEBUGGER, OnDebugger)
    ON_BN_CLICKED(ID_INTERRUPT_TRANSFER, OnInterruptTransfer)
    ON_BN_CLICKED(ID_PAUSE_TRANSFER, OnPauseTransfer)
    ON_BN_CLICKED(IDC_BUTTON_SETUP, OnButtonSetup)
  ON_MESSAGE(MYWM_NOTIFYICON, OnNotifyIcon)
    ON_BN_CLICKED(ID_CHECK_SMARTLOAD, OnCheckSmartload)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP();

  ////////////////////////////////////////////////////////////////////////////
  // CClientAppDlg message handlers

BOOL CClientAppDlg::OnInitDialog()
{
  CDialog::OnInitDialog();

 MyTaskBarAddIcon(m_hWnd, IDI_DOCICON16, m_hIcon16, "Atp Client");
```

```
  // Add "About..." menu item to system menu.

#ifdef  STUBBED_OUT
  // IDM_ABOUTBOX must be in the system command range.
  ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
  ASSERT(IDM_ABOUTBOX < 0xF000);

  CMenu* pSysMenu = GetSystemMenu(FALSE);
  CString strAboutMenu;
  strAboutMenu.LoadString(IDS_ABOUTBOX);
  if (!strAboutMenu.IsEmpty())
    {
      pSysMenu->AppendMenu(MF_SEPARATOR);
      pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
    }
#endif

  // Set the icon for this dialog.  The framework does this automatically
  //  when the application's main window is not a dialog
  SetIcon(m_hIcon32, TRUE);      // Set big icon
  SetIcon(m_hIcon16, FALSE);         // Set small icon

  // TODO: Add extra initialization here

    InitializeProtocolStuff();

    InitializeLoopTimers();

  return TRUE;  // return TRUE  unless you set the focus to a control
}

VOID CClientAppDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
  if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
      CAboutDlg dlgAbout;
      dlgAbout.DoModal();
    }
  else
    {
      CDialog::OnSysCommand(nID, lParam);
    }
}

VOID CClientAppDlg::OnDestroy()
{

// JRW Kill the timers ...
    KillLoopTimers();

  MyTaskBarDeleteIcon(m_hWnd, IDI_DOCICON16);
  WinHelp(0L, HELP_QUIT);
  CDialog::OnDestroy();
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

VOID CClientAppDlg::OnPaint()
```

```
{
  if (IsIconic())
    {
      CPaintDC dc(this); // device context for painting

      SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

      // Center icon in client rectangle
      int cxIcon = GetSystemMetrics(SM_CXICON);
      int cyIcon = GetSystemMetrics(SM_CYICON);
      CRect rect;
      GetClientRect(&rect);
      int x = (rect.Width() - cxIcon + 1) / 2;
      int y = (rect.Height() - cyIcon + 1) / 2;

      // Draw the icon
      dc.DrawIcon(x, y, m_hIcon32);
    }
  else
    {
    SetSecretDebugDialog();    // Set the secret debug mode.

    if (m_bStartHidden)      // Hack: real fix is to have a non-Modal dialog.
        {                    // ... this hack is to DoModal() showing the window
        m_bStartHidden = FALSE;
        ShowWindow(SW_HIDE);
        }
      CDialog::OnPaint();
    }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CClientAppDlg::OnQueryDragIcon()
{
  return (HCURSOR) m_hIcon16;
}

afx_msg LRESULT CClientAppDlg::OnNotifyIcon(WPARAM wParam, LPARAM lParam)
{
    UINT uID = (UINT) wParam;
    UINT uMouseMsg = (UINT) lParam;

    if ((uMouseMsg == WM_LBUTTONDOWN) || (uMouseMsg == WM_RBUTTONDOWN))
        {
        ShowWindow(SW_SHOW);
        LoopUpdateProgressDialog();  // Initialize progress bar for starters.
        }

    return 0;
}

VOID CClientAppDlg::OnHide()
{

    ShowWindow(SW_HIDE);
}


// Routine to get progress data ...
```

```
typedef struct tagPROGRESS_INFO
    {
    CString threadPriority;
    DWORD   ethernetCount;
    CString ethernetBreakdown;
    DWORD   averageEthernetLoad;
    DWORD   currentEthernetLoad;
    DWORD   protocolCount;
    CString protocolComparison;
    DWORD   averageProtocolLoad;
    DWORD   currentProtocolLoad;
    CString dialState;     // Return as a string for now ...
    long    fileAmount;
    long    fileSize;
    CString xferState;
    CString fileName;
    } PROGRESS_INFO;

// Macro-hack to make the code more readable ...
#define CHECK_RASCS_STATE(refstate) \
    if (rasconnstate ==  (refstate)) \
        { \
        pInfo->dialState = #refstate; \
        }

//JR: Basically, I need to by pass GetRASConnectionInformation
//when Rasapi32.dll is not available.

VOID CClientAppDlg::GetProgressValues(PVOID pProgressInfo)
{
HACKassert("Bad pProgressInfo pointer passed", pProgressInfo != NULL);

    PROGRESS_INFO * pInfo = (PROGRESS_INFO *) pProgressInfo;
    RASCONNSTATE rasconnstate;

// Get Connection status ...
    DWORD iRASConnectionCount; // == 0 if no RAS connection running.
    RASCONNSTATUS RasConnStatus;

    //Tim: If the RAS is not installed, m_hRasApi32Dll is null and
    //the next section is skipped.
    if(theApp.m_hRasApi32Dll != NULL)
        {
        if (!GetRASConnectionInformation
            (theApp.m_hRasApi32Dll, &g_DialState, iRASConnectionCount, RasConnStatus))
            {
HACKassert("XXX Cannot get connection information",FALSE);
            }
        rasconnstate = RasConnStatus.rasconnstate;

// Get string for connection state ...
            CHECK_RASCS_STATE(RASCS_OpenPort)
        else CHECK_RASCS_STATE(RASCS_PortOpened)
        else CHECK_RASCS_STATE(RASCS_ConnectDevice)
        else CHECK_RASCS_STATE(RASCS_DeviceConnected)
        else CHECK_RASCS_STATE(RASCS_AllDevicesConnected)
        else CHECK_RASCS_STATE(RASCS_Authenticate)
```

```
        else CHECK_RASCS_STATE(RASCS_AuthNotify)
        else CHECK_RASCS_STATE(RASCS_AuthRetry)
        else CHECK_RASCS_STATE(RASCS_AuthCallback)
        else CHECK_RASCS_STATE(RASCS_AuthChangePassword)
        else CHECK_RASCS_STATE(RASCS_AuthProject)
        else CHECK_RASCS_STATE(RASCS_AuthLinkSpeed)
        else CHECK_RASCS_STATE(RASCS_AuthAck)
        else CHECK_RASCS_STATE(RASCS_ReAuthenticate)
        else CHECK_RASCS_STATE(RASCS_Authenticated)
        else CHECK_RASCS_STATE(RASCS_PrepareForCallback)
        else CHECK_RASCS_STATE(RASCS_WaitForModemReset)
        else CHECK_RASCS_STATE(RASCS_WaitForCallback)
        else CHECK_RASCS_STATE(RASCS_Projected)
        else CHECK_RASCS_STATE(RASCS_SubEntryConnected)
        else CHECK_RASCS_STATE(RASCS_SubEntryDisconnected)
        else CHECK_RASCS_STATE(RASCS_Interactive)
        else CHECK_RASCS_STATE(RASCS_RetryAuthentication)
        else CHECK_RASCS_STATE(RASCS_CallbackSetByCaller)
        else CHECK_RASCS_STATE(RASCS_PasswordExpired)
        else CHECK_RASCS_STATE(RASCS_Connected)
        else CHECK_RASCS_STATE(RASCS_Disconnected)
        else{
            CString msg;
            msg.Format( "Unknown RASCS state: %ld", long(rasconnstate));
    HACKassert(msg,FALSE);
            }
        }
    else
        { //For the case when no RAS is installed.
        pInfo->threadPriority = _T("");
        pInfo->ethernetCount = 0;
        pInfo->ethernetBreakdown = _T("");
        pInfo->averageEthernetLoad = 0;
        pInfo->currentEthernetLoad = 0;
        pInfo->protocolCount = 0;
        pInfo->protocolComparison = _T("");
        pInfo->averageProtocolLoad = 0;
        pInfo->currentProtocolLoad = 0;
        pInfo->dialState = _T("");
        pInfo->fileAmount = 0;
        pInfo->fileSize = 0;
        pInfo->xferState = _T("");
        pInfo->fileName = _T("");
        }

// XXX Should be accessed with functions ...
    if (m_bSecretDebugMode)
        {
        if      (UGLYThreadPriority==THREAD_PRIORITY_HIGHEST)
            pInfo->threadPriority="HIGHEST";
        else if (UGLYThreadPriority==THREAD_PRIORITY_ABOVE_NORMAL)
            pInfo->threadPriority="ABOVE_NORMAL";
        else if (UGLYThreadPriority==THREAD_PRIORITY_NORMAL)
            pInfo->threadPriority="NORMAL";
        else if (UGLYThreadPriority==THREAD_PRIORITY_BELOW_NORMAL)
            pInfo->threadPriority="BELOW_NORMAL";
        else if (UGLYThreadPriority==THREAD_PRIORITY_LOWEST)
            pInfo->threadPriority="LOWEST";
        else if (UGLYThreadPriority==THREAD_PRIORITY_IDLE)
            pInfo->threadPriority="IDLE";
```

```
        else{
HACKassert("Invalid thread priority computed!",FALSE);
            pInfo->threadPriority  = "<internal error>";
            }

        pInfo->ethernetCount          = UGLYEthernetCount;
        pInfo->ethernetBreakdown.Format("%10lu, %10lu",
                              UGLYEthernetReceiveCount,
                              UGLYEthernetSendCount);
        pInfo->averageEthernetLoad    = UGLYaverageEthernetLoad;
        pInfo->currentEthernetLoad    = UGLYcurrentEthernetLoad;
        pInfo->protocolCount          = UGLYProtocolCount;

        UGLYMutex.Lock();
        pInfo->protocolComparison     = UGLYProtocolComparison;
        UGLYMutex.Unlock();

        pInfo->averageProtocolLoad    = UGLYaverageProtocolLoad;
        pInfo->currentProtocolLoad    = UGLYcurrentProtocolLoad;
        }

#ifdef  STUBBED_OUT
static long fakeFileAmount  = 0;
static long fakeFileSize    = 7000000L;
    if (fakeFileAmount >= fakeFileSize)
        {
        fileAmount = fileSize = 0;
        fakeFileAmount = 0;        // For next time.
        return;
        }
    fakeFileAmount += (fakeFileSize / 12);
    if (fakeFileAmount > fakeFileSize)
        fakeFileAmount = fakeFileSize;
    fileAmount  = fakeFileAmount;
    fileSize    = fakeFileSize;
    xferState = "defrauded";
    fileName    = "pudn'n tane";
#endif

    pInfo->fileAmount = pInfo->fileSize = 0;
    pInfo->fileName = "<no file>";
    pInfo->xferState = "<unknown: error?>";

assert(("Bad internal pointer m_atpserver",m_atpserver != NULL));
    const CPtrArray * p_conns = m_atpserver->GetAllConnectionStates();
assert(("Bad internal pointer p_conns",p_conns != NULL));
    int connectionCount = p_conns->GetSize();
    if (connectionCount == 0)
        {
        pInfo->xferState = "No connections in progress";
        pInfo->fileAmount  = pInfo->fileSize = 0;
        pInfo->fileName    = "";
        return;
        }
// Note:
// Multiple connections are a by-product of sockets not being really closed
// until TCP/IP gets around to closing them asynchronously.

    int connectionIdx = connectionCount -1;
    AtpConnection::AtpState atpState =
```

```
                ((AtpConnection*)(p_conns->GetAt(connectionIdx)))->state;
    if      (atpState==AtpConnection::CONNECTING) pInfo->xferState="CONNECTING";
    else if (atpState==AtpConnection::WAITING)    pInfo->xferState="WAITING";
    else if (atpState==AtpConnection::RECEIVING)  pInfo->xferState="RECEIVING";
    else if (atpState==AtpConnection::CLOSING)    pInfo->xferState="CLOSING";
    else if (atpState==AtpConnection::PAUSED)     pInfo->xferState="PAUSED";

    if ((atpState == AtpConnection::RECEIVING) ||
        (atpState == AtpConnection::PAUSED))
        {
        IDFile * idf =((AtpConnection*)(p_conns->GetAt(connectionIdx)))->idf;
        pInfo->fileAmount    = idf->GetLoadedSize();
        pInfo->fileSize      = idf->GetSize();
        pInfo->fileName      = idf->GetFileName();
        }

}

// HACK: Try to reduce the flicker in the things we display
// in the dialog ...

CString g_tempUpdateWindowText;
VOID UpdateWindowText(CWnd & window, const CString & newText)
    {
    if (window.m_hWnd == NULL)  // HACK: Callback to update window called
        return;                 // after window destroyed: not a good test,
                                // the threads are asynchronous.
    window.GetWindowText(g_tempUpdateWindowText);
    if (g_tempUpdateWindowText != newText)
        window.SetWindowText(newText);
    }

// Update progress-bar dialog ...

VOID CClientAppDlg::LoopUpdateProgressDialog()
{
// TGT: do not update unless the window has been initialized
    if (m_hWnd == NULL)      // Theo, I still think this might be the
        return;              // wrong way to do this!

    PROGRESS_INFO State;

    GetProgressValues((PVOID) (&State));
    if (State.fileSize < 0)
        {
        CString msg;
        msg.Format("State.fileSize for transfer bad: %ld",long(State.fileSize));
HACKassert(msg,State.fileSize >= 0);
        }
    if (State.fileAmount < 0)
        {
        CString msg;
        msg.Format("State.fileAmount for transfer bad: %ld",
                long(State.fileAmount));
HACKassert(msg,State.fileAmount >= 0);
        }
    CString progressString1(State.xferState);
    CString progressString2("                                                    ");
    if (State.fileSize == 0)  // Nothing transfering right now.
        {
```

```
        m_PROGRESS_BAR.SetPos(0);
        }
    else{                   //; Something transferring.
        if (State.fileAmount > State.fileSize)
            {
            CString msg;
            msg.Format("State.fileAmount not <= State.fileSize, %ld <= %ld",
                long(State.fileAmount), long(State.fileSize));
HACKassert(msg,State.fileAmount >= 0);
            }
        int iPerCent = int(MulDiv(State.fileAmount,100,State.fileSize));
        m_PROGRESS_BAR.SetPos(iPerCent);
        progressString1 += ": " + State.fileName;
        progressString2.Format("Transmitted %8ld of %ld                ",
                long(State.fileAmount), long(State.fileSize));
        }
    UpdateWindowText(m_ProgressLabel1, progressString1);
    UpdateWindowText(m_ProgressLabel2, progressString2);

// Debug display of dial state, thread priority ...

    UpdateWindowText(m_ProgressLabel3, "RAS Dial state: "
                                        + State.dialState);
    UpdateWindowText(m_ProgressLabel4, "Protocol thread priority: "
                                        + State.threadPriority);

// Debug display of network measurements ...

    CString tTemp;

    tTemp.Format("Ethernet total bytes: %11ld", (long) State.ethernetCount);
    UpdateWindowText(m_ProgressLabel5, tTemp);
    tTemp.Format("... Send/Receive: %s", State.ethernetBreakdown);
    UpdateWindowText(m_ProgressLabel6, tTemp);
    tTemp.Format("... most recent: %9ld, average: %9ld bytes/second",
                (long) State.currentEthernetLoad,
                (long) State.averageEthernetLoad);
    UpdateWindowText(m_ProgressLabel7, tTemp);

    tTemp.Format("Protocol total bytes: %11ld", (long) State.protocolCount);
    UpdateWindowText(m_ProgressLabel8, tTemp);
    tTemp.Format("... most recent: %9ld, average: %9ld bytes/second",
                (long) State.currentProtocolLoad,
                (long) State.averageProtocolLoad);
    UpdateWindowText(m_ProgressLabel9, tTemp);
    tTemp.Format("Comparison: %s", State.protocolComparison);
    UpdateWindowText(m_ProgressLabel10, tTemp);

}

VOID CClientAppDlg::LoopCheckLoadRequests()
{
    BOOL bKilled = KillTimer(m_IDTimerLoadRequest);
HACKassert("Cannot Kill Timer on m_IDTimerLoadRequest",bKilled);

    CheckAllSmartLoadRequests(m_atpserver, m_idfserver, m_configVals);
    int reqTime = AtpAttrib::ParseInt(m_configVals["LoadRequestPeriod"]);
    if (reqTime > 0)
        {
        m_IDTimerLoadRequest =
```

```
                SetTimer(EVENT_LOAD_REQUEST, reqTime*1000L, NULL);
HACKassert("Cannot SetTimer on m_IDTimerLoadRequest",m_IDTimerLoadRequest != 0);
        }
}


VOID CClientAppDlg::LoopClientPollServer()
{
// Check in with the designated server ...
    m_atpserver->SetRequestTransfer(0); // Continue transfers.

    AtpAttrib providers;
    providers.ParseLine(m_configVals["Providers"]);

    theApp.DialRASConnectionAutomatically();
    m_atpserver->PollServers(providers);
}

VOID CClientAppDlg::OnTimer(UINT nIDEvent)
{
    // TODO: Add your message handler code here and/or call default

    CDialog::OnTimer(nIDEvent);

    if (nIDEvent == EVENT_PROGRESS)     // Progress Bar.
        {
        BOOL bKilled = KillTimer(m_IDTimerProgress);
HACKassert("Cannot Kill Timer on m_IDTimerProgress",bKilled);

        LoopUpdateProgressDialog();

        int updateTime = AtpAttrib::ParseInt(m_configVals["StatusPeriod"]);
        m_IDTimerProgress = SetTimer(EVENT_PROGRESS, updateTime, NULL);
HACKassert("Cannot SetTimer on m_IDTimerProgress",m_IDTimerProgress != 0);
        }

    if (nIDEvent == EVENT_POLL_SERVER)     // Check schedule.
        {
        BOOL bKilled = KillTimer(m_IDTimerPollServer);// No double-timer firing.
HACKassert("Cannot Kill Timer on m_IDTimerPollServer",bKilled);

        LoopClientPollServer(); // Check in with the designated server ...

        int pollTime = AtpAttrib::ParseInt(m_configVals["ClientPollingPeriod"]);
        if (pollTime > 0)
            {
            m_IDTimerPollServer =
                    SetTimer(EVENT_POLL_SERVER, pollTime*1000L, NULL);
HACKassert("Cannot SetTimer on m_IDTimerPollServer",m_IDTimerPollServer != 0);
            }
        }

    if (nIDEvent == EVENT_LOAD_REQUEST)     // Check for SmartLoad requests
        {
        LoopCheckLoadRequests(); // Check in with the designated server ...

        }

}
```

```
VOID CClientAppDlg::OnClose()
{
    // TODO: Add your message handler code here and/or call default

    CDialog::OnClose();
}

// Callback routine: called by file transfer when the state
// of a connection has changed.
// For the moment, we don't used the value passed,
// we just poll to get the whole state information separately
// and only use this routine to tell us that we need to check something.

VOID CClientAppDlg:: ProcessConnectionState(const AtpConnection& conn)
{
    //(PVOID) &conn;  // To suppress compiler warning

    if (conn.state == AtpConnection::PAUSED)
        LoopUpdateProgressDialog();     // To update dialog ...
}


int CClientAppDlg::DoModal()     // To get state.
{
    // TODO: Add your specialized code here and/or call the base class
    return CDialog::DoModal();

}



VOID CClientAppDlg::OnConfigure()
{
    CDbgConfigDlg dlgDbgConfig;

    int nResultThing = dlgDbgConfig.DoModal();

HACKassert("Dummy assert", nResultThing == nResultThing);

}

// Set height and width of dialog to hide/show the debugging stuff
// we have below the "public" part of the dialog on top ...

VOID CClientAppDlg::SetSecretDebugDialog()
{
    CRect mainRect;
    GetWindowRect(&mainRect);  // Returns screen coordinates
    int newHeight = mainRect.Height();
    int newWidth  = mainRect.Width();
    CPoint mainTopLeft = mainRect.TopLeft();
    if (m_bSecretDebugMode)
        {
        CRect bottomDebugRect;
        m_SECRET_DEBUG_TEXT_BOTTOM.GetWindowRect(&bottomDebugRect);
        CPoint bottomDebugBottomRight = bottomDebugRect.BottomRight();
        newHeight = bottomDebugBottomRight.y - mainTopLeft.y + 10;
        newWidth  = bottomDebugBottomRight.x - mainTopLeft.x + 10;
```

```
        }
    else{
        CRect topDebugRect;
        m_SECRET_DEBUG_TEXT_TOP.GetWindowRect(&topDebugRect);
        CPoint topDebugTopLeft     = topDebugRect.TopLeft();
        CPoint topDebugBottomRight = topDebugRect.BottomRight();
        newHeight = topDebugTopLeft.y     - mainTopLeft.y - 2;
        newWidth  = topDebugBottomRight.x - mainTopLeft.x + 10;
        }
    SetWindowPos(
            NULL,        // New Z order
            0, 0,        // New position to move to
            newWidth, newHeight,
            SWP_NOZORDER | SWP_NOMOVE);
}


// Check for a double-right-click on the little "company icon" image in
// the upper-left of the dialog.

VOID CClientAppDlg::OnRButtonDblClk(UINT nFlags, CPoint point)
{
// Get rect for our secret icon, and convert to client co-ordinates for the
// main dialog, to match the mouse point location:

    CRect iconRect;
    m_ICON_IMAGE.GetWindowRect(&iconRect);  // Returns screen coordinates
    ScreenToClient(&iconRect);      // Convert to dialog's client co-ordinates

// Now that the mouse point and icon rect are in the same co-ordinate system,
// check whether the user clicked in the icon:

    BOOL bInside = iconRect.PtInRect(point);

// If they clicked inside, and they are holding down both the
// control and the shift keys, bring up our secret feature ...

    if (bInside && (nFlags & MK_CONTROL) && (nFlags & MK_SHIFT))
        {
        m_bSecretDebugMode = !m_bSecretDebugMode;
// If in debug mode, hide the "secret" stuff in the dialog which is below
// the progress bar ...
        SetSecretDebugDialog();
        }

// "Now we return to your previously scheduled programming, in progress ..."

    CDialog::OnRButtonDblClk(nFlags, point);
}


VOID CClientAppDlg::OnButtonAbout()
{
    CAboutDlg dlgAbout;
    dlgAbout.DoModal();

}

VOID CClientAppDlg::OnButtonHelp()
{
```

```
    MessageBox("Help for this dialog is not yet implemented:\n"
                "Sorry!",
                "Beta-Release Message " + g_ClientAppBuildDate,
                MB_OK | MB_ICONEXCLAMATION);

}


VOID CClientAppDlg::OnDebugger()
{
    DebugBreak();        // Just force a break to the debugger.
}

VOID CClientAppDlg::OnInterruptTransfer()
{

// This is for debugging checks only ...
    m_atpserver->SetRequestTransfer(0); // Continue transfers.
    m_atpserver->SetRequestTransfer(-1);   // Interrupt.
// -1 means cut connection, don't restart.

}
VOID CClientAppDlg::OnPauseTransfer()
{
// This is for debugging checks only ...
    m_atpserver->SetRequestTransfer(0); // Continue transfers.
    m_atpserver->SetRequestTransfer(30);   // Delay thirty seconds.
// -1 means cut connection, don't restart.


}

VOID CClientAppDlg::InitializeProtocolStuff()
{
    m_atpserver = new SocketControl(m_configVals);   // ClientSocketControl
    m_contentserver = new ContentControl(&m_configVals);    // TGT 961130
    m_idfserver = new IDFServer(m_contentserver, &m_configVals);

// JRW:
// Initialize needs access to state-has-changed callback function in our dialog.
   if (!m_atpserver->Initialize(m_idfserver, this))
        {
// NOTE: TGT says ok to continue, even though socket is gronked:
// effect is just that we can't initiate communications from the server,
// because the client won't be listening on the socket ...
// however, we can still poll, so we probably shouldn't tell the user
// to reboot yet (so says Theo), they will reboot eventually anyway.

HACKassert("Cannot bind to socket: you may continue anyway ...",FALSE);
// NOTE: We need to enforce client initiated connections in this case (TGT)
        }

// Start monitor thread ...

    if (!StartMonitorThread(m_atpserver, m_configVals))
        {
        AfxMessageBox("Cannot start monitor thread!");
        }
}
```

```
VOID CClientAppDlg::UnInitializeProtocolStuff()
{
    m_atpserver->CloseConnections();     // TGT 961130

    KillMonitorThread();
}

CClientAppDlg::~CClientAppDlg()    // Destructor.
    {
    delete m_atpserver;
    delete m_contentserver; // JRW 961203
    delete m_idfserver;
    }

VOID CClientAppDlg::OnButtonSetup()
{
    CConfigureDlg dlgConfigure(m_configVals);

    int nResultThing = dlgConfigure.DoModal();

    if (nResultThing == IDOK)
        {           // Need to re-set polling times now ...
        KillLoopTimers();
        InitializeLoopTimers();
        }

}

VOID CClientAppDlg::OnButtonClientPollServerNow()
{
    LoopClientPollServer(); // Check in with the designated server ...
}


void CClientAppDlg::OnCheckSmartload()
{
    LoopCheckLoadRequests();

}

#ifdef  SOURCE_CONTROL_BLOCK
 Project Name: E-PS Client/Server program
 E-PS Inc.
 Copyright 1996, 1997. All Rights Reserved.
 SUBSYSTEM: ClientApp customer client program
 $Workfile: ClientAppDlg.cpp $
 $Author: Tding $
 $Revision: 47 $
 $Date: 2/03/97 10:24a $
 $Modtime: 2/03/97 10:23a $

 $History: ClientAppDlg.cpp $
 *
 * *****************  Version 47  *****************
 * User: Tding        Date: 2/03/97    Time: 10:24a
 * Updated in $/atp/ClientApp
 * Changes made for no RAS situation.
 *
 * *****************  Version 46  *****************
 * User: Jrward       Date: 1/31/97    Time: 2:48p
```

```
* Updated in $/atp/ClientApp
* Global check-in.
*
* *****************  Version 45  *****************
* User: Jrward        Date: 1/31/97     Time: 10:47a
* Updated in $/atp/ClientApp
* Stubbed out some netstat asserts for now.
*
* *****************  Version 44  *****************
* User: Jrward        Date: 1/30/97     Time: 1:48p
* Updated in $/atp/ClientApp
* Check everything I have in, so Theo can get it.
*
* *****************  Version 43  *****************
* User: Jrward        Date: 1/29/97     Time: 5:30p
* Updated in $/atp/ClientApp
* End-Of-Day checkin festival.
*
* *****************  Version 42  *****************
* User: Jrward        Date: 1/28/97     Time: 1:11p
* Updated in $/atp/ClientApp
* Afternoon check-in -- mostly fixes to installation from all the
* SmartLoad fixes.
*
* *****************  Version 41  *****************
* User: Jrward        Date: 1/27/97     Time: 5:11p
* Updated in $/atp/ClientApp
* SmartLoad sort of working ...
*
* *****************  Version 40  *****************
* User: Jrward        Date: 1/27/97     Time: 11:01a
* Updated in $/atp/ClientApp
* Smartload stuff moved to separate source files smartload.*
*
* *****************  Version 39  *****************
* User: Jrward        Date: 1/24/97     Time: 4:58p
* Updated in $/atp/ClientApp
* New .epsclient SmartLoad file handling: Just waiting on Theo's stuff,
* which he hasn't tested yet.
*
* *****************  Version 38  *****************
* User: Jrward        Date: 1/23/97     Time: 4:50p
* Updated in $/atp/ClientApp
* Daily check-in: looks like I have the SmartLoad stuff working, I just
* need the new Client-side protocol code from Theo.
*
* *****************  Version 37  *****************
* User: Jrward        Date: 1/23/97     Time: 3:04p
* Updated in $/atp/ClientApp
* SmartLoad request files handled at the file level (*.epsclient in
* LoadRequestDir directory), but no processing of the innards of the
* files yet.
*
* *****************  Version 36  *****************
* User: Jrward        Date: 1/23/97     Time: 12:26p
* Updated in $/atp/ClientApp
* Added /LoadRequest switch on clientapp.exe, so can use it as "helper"
* app with Web browsers for SmartLoad.  Actual SmartLoad processing not
* done yet.
*
```

```
* ******************  Version 34  ******************
* User: Jrward        Date: 1/20/97    Time: 12:19p
* Updated in $/atp/ClientApp
* Put a mutex around some of the "UGLY" stuff in our debugging display:
* we were crashing.
*
* ******************  Version 33  ******************
* User: Jrward        Date: 1/13/97    Time: 5:02p
* Updated in $/atp/ClientApp
* End-of-the-day checkin.
*
* ******************  Version 32  ******************
* User: Jrward        Date: 1/13/97    Time: 5:00p
* Updated in $/atp/ClientApp
* Cannot get ODBC calls in fake JRDemo.exe to work from the
* ServerISAPI.DLL: work fine when running www server from debugger, but
* throw when opening the CustomerDB.mdb database when running normally as
* a service!
*
* ******************  Version 31  ******************
* User: Jrward        Date: 1/13/97    Time: 9:36a
* Updated in $/atp/ClientApp
* Small change in "Setup" dialog for setting IDF directory: still more to
* do.
*
* ******************  Version 30  ******************
* User: Jrward        Date: 1/07/97    Time: 12:16a
* Updated in $/atp/ClientApp
* StatusPeriod now configurable.
*
* ******************  Version 29  ******************
* User: Ttonchev      Date: 1/06/97    Time: 11:19p
* Updated in $/atp/ClientApp
* Cleared update at block receive
*
* ******************  Version 28  ******************
* User: Jrward        Date: 1/06/97    Time: 1:23p
* Updated in $/atp/ClientApp
* Fixed the SetSecret... stuff for hidding our debugging buttons, added
* OnButtonSetup
*
* ******************  Version 27  ******************
* User: Jrward        Date: 1/03/97    Time: 12:56p
* Updated in $/atp/ClientApp
* Small fixes in debugging display dialog.
*
* ******************  Version 26  ******************
* User: Jrward        Date: 1/02/97    Time: 5:15p
* Updated in $/atp/ClientApp
* End of the day check-in.
*
* ******************  Version 25  ******************
* User: Jrward        Date: 1/02/97    Time: 10:57a
* Updated in $/atp/ClientApp
* Added SOURCE_CONTROL_BLOCK stuff.
//-
// JRW 961128   Added EVENT_SCHEDULE_CHECK timer code: but no guts yet.
//              Changed to use updated AtpAttrib class for configuration values
// JRW 961203   Small fixes.
// JRW 961213   Hack to make window hidden on startup:
```

```
//                The window flashes briefly the way we do it here, the
//                real fix would be to have a non-Modal dialog: DoModal
//                always makes the window show up.
// JRW 961215     Moved "connect to server at startup" to pollling loop in
//                ClientAppDlg from ClientApp: it was too hard getting the
//                non-Window SetTimer functions to work with the application
//                class.
// JRW 961220     Moved protocol initialization, shut-down from ClientApp.cpp
//                to ClientAppDlg.cpp, to fix the problem of the update
//                call-back function being called after the main dialog was
//                already destroyed:
//                    m_idfserver, m_atpserver, m_contentserver
#endif     // SOURCE_CONTROL_BLOCK
```

```
// ClientAppDlg.h : header file
//
#ifndef CLIENTAPPDLG_H
#define CLIENTAPPDLG_H

/////////////////////////////////////////////////////////////////////////////
// CClientAppDlg dialog

class CClientAppDlg : public CDialog, public IAtpProgressCallback
{
// Construction
public:
    CClientAppDlg(AtpAttrib & configVals, CWnd* pParent = NULL);
                // standard constructor
  ~CClientAppDlg();  // Destructor.

    virtual VOID ProcessConnectionState(const AtpConnection& conn);

// Dialog Data
    //{{AFX_DATA(CClientAppDlg)
    enum { IDD = IDD_CLIENT_DIALOG };
    CStatic m_ProgressLabel10;
    CStatic m_SECRET_DEBUG_TEXT_TOP;
    CStatic m_SECRET_DEBUG_TEXT_BOTTOM;
    CStatic m_ICON_IMAGE;
    CStatic m_ProgressLabel9;
    CStatic m_ProgressLabel8;
    CStatic m_ProgressLabel7;
    CStatic m_ProgressLabel6;
    CStatic m_ProgressLabel5;
    CStatic m_ProgressLabel4;
    CStatic m_ProgressLabel3;
    CStatic m_ProgressLabel2;
    CStatic m_ProgressLabel1;
    CProgressCtrl   m_PROGRESS_BAR;
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CClientAppDlg)
    public:
    virtual int DoModal();
    protected:
    virtual VOID DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    HICON m_hIcon16;
    HICON m_hIcon32;
    UINT    m_IDTimerProgress;          // ID of timer for the progress-bar
    UINT    m_IDTimerPollServer;        // ID for polling server.
    UINT    m_IDTimerLoadRequest;       // ID for check-smartload-directory

    // Generated message map functions
    //{{AFX_MSG(CClientAppDlg)
    virtual BOOL OnInitDialog();
    afx_msg VOID OnSysCommand(UINT nID, LPARAM lParam);
  - afx_msg VOID OnDestroy();
    afx_msg VOID OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
```

1

```
    afx_msg VOID OnHide();
    afx_msg VOID OnTimer(UINT nIDEvent);
    afx_msg VOID OnClose();
    afx_msg VOID OnConfigure();
    afx_msg VOID OnRButtonDblClk(UINT nFlags, CPoint point);
    afx_msg VOID OnButtonAbout();
    afx_msg VOID OnButtonHelp();
    afx_msg VOID OnButtonClientPollServerNow();
    afx_msg VOID OnDebugger();
    afx_msg VOID OnInterruptTransfer();
    afx_msg VOID OnPauseTransfer();
    afx_msg VOID OnButtonSetup();
    afx_msg void OnCheckSmartload();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP();

    afx_msg LRESULT OnNotifyIcon(WPARAM wParam, LPARAM lParam);

private:
// Theo's stuff ...
    AtpAttrib & m_configVals;
    IDFServer *      m_idfserver;
    SocketControl * m_atpserver;
    ContentControl *_m_contentserver;

    BOOL m_bStartHidden;
    BOOL m_bSecretDebugMode;        // JRW Show "secret debugging" part of dialog
#ifdef  OLD_STUFF    // moved to SocketControl
    BOOL ServerConnectionAlready(const CString & serverName);
#endif  OLD_STUFF
    VOID KillLoopTimers();
    VOID InitializeLoopTimers();

    VOID LoopClientPollServer();

    VOID LoopCheckLoadRequests();

    VOID LoopUpdateProgressDialog();    // To update progress bar ...
    VOID GetProgressValues(PVOID pProgressInfo);

    VOID SetSecretDebugDialog();
    VOID InitializeProtocolStuff();
    VOID UnInitializeProtocolStuff();

};

#ifdef  SOURCE_CONTROL_BLOCK
 Project Name: E-PS Client/Server program
 E-PS Inc.
 Copyright 1996, 1997. All Rights Reserved.
 SUBSYSTEM: ClientApp customer client program
 $Workfile: ClientAppDlg.h $
 $Author: Jrward $
 $Revision: 37 $
 $Date: 1/31/97 2:48p $
 $Modtime: 1/25/97 7:58p $

 $History: ClientAppDlg.h $
 *
 * *****************  Version 37  *****************
```

```
* User: Jrward        Date: 1/31/97     Time: 2:48p
* Updated in $/atp/ClientApp
* Global check-in.
*
* *****************  Version 36  *****************
* User: Jrward        Date: 1/31/97     Time: 10:47a
* Updated in $/atp/ClientApp
* Stubbed out some netstat asserts for now.
*
* *****************  Version 35  *****************
* User: Jrward        Date: 1/30/97     Time: 1:48p
* Updated in $/atp/ClientApp
* Check everything I have in, so Theo can get it.
*
* *****************  Version 34  *****************
* User: Jrward        Date: 1/29/97     Time: 5:30p
* Updated in $/atp/ClientApp
* End-Of-Day checkin festival.
*
* *****************  Version 33  *****************
* User: Jrward        Date: 1/28/97     Time: 1:11p
* Updated in $/atp/ClientApp
* Afternoon check-in -- mostly fixes to installation from all the
* SmartLoad fixes.
*
* *****************  Version 32  *****************
* User: Jrward        Date: 1/27/97     Time: 5:11p
* Updated in $/atp/ClientApp
* SmartLoad sort of working ...
*
* *****************  Version 31  *****************
* User: Jrward        Date: 1/27/97     Time: 10:29a
* Updated in $/atp/ClientApp
* Smartload stuff now in separate sources SmartLoad.cpp, SmartLoad.h
*
* *****************  Version 30  *****************
* User: Jrward        Date: 1/24/97     Time: 4:58p
* Updated in $/atp/ClientApp
* Added button for "Check Smartload Now"
*
* *****************  Version 29  *****************
* User: Jrward        Date: 1/23/97     Time: 4:50p
* Updated in $/atp/ClientApp
* Daily check-in: looks like I have the SmartLoad stuff working, I just
* need the new Client-side protocol code from Theo.
*
* *****************  Version 28  *****************
* User: Jrward        Date: 1/23/97     Time: 3:04p
* Updated in $/atp/ClientApp
* SmartLoad request files handled at the file level (*.epsclient in
* LoadRequestDir directory), but no processing of the innards of the
* files yet.
*
* *****************  Version 27  *****************
* User: Jrward        Date: 1/23/97     Time: 12:26p
* Updated in $/atp/ClientApp
* Added /LoadRequest switch on clientapp.exe, so can use it as "helper"
* app with Web browsers for SmartLoad.  Actual SmartLoad processing not
* done yet.
*
```

```
* *****************  Version 26  *****************
* User: Jrward        Date: 1/20/97    Time: 12:19p
* Updated in $/atp/ClientApp
* Put a mutex around some of the "UGLY" stuff in our debugging display:
* we were crashing.
*
* *****************  Version 25  *****************
* User: Jrward        Date: 1/13/97    Time: 5:02p
* Updated in $/atp/ClientApp
* End-of-the-day checkin.
*
* *****************  Version 24  *****************
* User: Jrward        Date: 1/13/97    Time: 5:00p
* Updated in $/atp/ClientApp
* Cannot get ODBC calls in fake JRDemo.exe to work from the
* ServerISAPI.DLL: work fine when running www server from debugger, but
* throw when opening the CustomerDB.mdb database when running normally as
* a service!
*
* *****************  Version 23  *****************
* User: Jrward        Date: 1/13/97    Time: 9:36a
* Updated in $/atp/ClientApp
* Small change in "Setup" dialog for setting IDF directory: still more to
* do.
*
* *****************  Version 22  *****************
* User: Jrward        Date: 1/07/97    Time: 12:16a
* Updated in $/atp/ClientApp
* StatusPeriod now configurable.
*
* *****************  Version 21  *****************
* User: Ttonchev      Date: 1/06/97    Time: 11:20p
* Updated in $/atp/ClientApp
*
* *****************  Version 20  *****************
* User: Jrward        Date: 1/06/97    Time: 1:23p
* Updated in $/atp/ClientApp
* Fixed the SetSecret... stuff for hidding our debugging buttons, added
* OnButtonSetup
*
* *****************  Version 18  *****************
* User: Jrward        Date: 1/02/97    Time: 10:57a
* Updated in $/atp/ClientApp
* Added SOURCE_CONTROL_BLOCK stuff.

// JRW 961127 Added stuff for EVENT_SCHEDULE_CHECK Timer function.
// JRW 961215   Moved "connect to server at startup" to polling loop in
//              ClientAppDlg from ClientApp: it was too hard getting the
//              non-Window SetTimer functions to work with the application
//              class.
// JRW 961217   m_updateMutex, other stuff.
// JRW 961220   m_updateMutex et al remove, now that all engineers agree
//              this was not the correct solution.
// JRW 961220   Moved protocol initialization, shut-down from ClientApp.cpp
//              to ClientAppDlg.cpp, to fix the problem of the update
//              call-back function being called after the main dialog was
//              already destroyed:
//                  m_idfserver, m_atpserver, m_contentserver
#endif    // SOURCE_CONTROL_BLOCK

#endif  // CLIENTAPPDLG_H
```

```
// atpmemfile.cpp: implementation of class AtpMemFile

#include "stdafx.h"
#include "atpmemfile.h"
#include "atpdata.h"
#include "atpexcept.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

AtpMemFile::AtpMemFile(AtpAttrib& attrib) : AtpFile() {
  SetAttributes(attrib);
  BindToFile(new CMemFile());
}

AtpMemFile::~AtpMemFile() {
  delete GetFile();
}

void AtpMemFile::GetAttributes(AtpAttrib& attrib) {
  AtpAttrib arg;
  AtpFile::GetAttributes(arg);
  long size;
  try {
    size = AtpAttrib::ParseInt(arg[ATP_ARG_SIZE]);
  } catch (FormatException ex) {
    ex;
    size = 0;
  }

  long fsize = GetFile()->GetLength();
  if (fsize != size) {
    arg[ATP_ARG_SIZE] = AtpAttrib::UnparseInt(fsize);
    SetAttributes(arg);
  }

  attrib = arg;
}

#ifdef  SOURCE_CONTROL_BLOCK
 Project Name: E-PS Client/Server program
 E-PS Inc.
 Copyright 1996, 1997. All Rights Reserved.
 SUBSYSTEM: EPS Protocol
 $Workfile: atpmemfile.cpp $
 $Author: Jrward $
 $Revision: 3 $
 $Date: 1/27/97 12:07p $
 $Modtime: 1/27/97 12:06p $

 $History: atpmemfile.cpp $
 *
 * *****************  Version 3  *****************
 * User: Jrward       Date: 1/27/97    Time: 12:07p
 * Updated in $/atp/Protocol
 * Added VSS history stuff.
#endif   // SOURCE_CONTROL_BLOCK
```

```
//
// C++ header file
// (c) 1996 ACS
//

#ifndef _ATPMEMFILE_H_
#define _ATPMEMFILE_H_

#include "stdafx.h"
#include "atpfile.h"

class AtpMemFile : public AtpFile {
private:
  // disallow copy contructor and assignment
  AtpMemFile(const AtpFile&);
  void operator=(const AtpMemFile&);

public:
  AtpMemFile(AtpAttrib& attrib);
  virtual ~AtpMemFile();

  virtual void GetAttributes(AtpAttrib& attrib);
};


#endif // _ATPMEMFILE_H_

// End of headers


#ifdef  SOURCE_CONTROL_BLOCK
 Project Name: E-PS Client/Server program
 E-PS Inc.
 Copyright 1996, 1997. All Rights Reserved.
 SUBSYSTEM: E-parcel.com
 $Workfile: atpmemfile.h $
 $Author: Jrward $
 $Revision: 2 $
 $Date: 1/27/97 12:13p $
 $Modtime: 1/27/97 12:11p $

 $History: atpmemfile.h $
 *
 * *****************  Version 2  *****************
 * User: Jrward       Date: 1/27/97    Time: 12:13p
 * Updated in $/atp/Include
 * Added VSS history stuff.
#endif    // SOURCE_CONTROL_BLOCK
```

```
// atpserver.cpp : implementation of the AtpServerSocket class

#include <stdlib.h>
#include "stdafx.h"
#include <assert.h>

#include "atpserver.h"
#include "atpexcept.h"
#include "atpdata.h"
#include "atpmemfile.h"
#include "atprepfile.h"
#include "utility.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#define CONFIGFILEDIR "\\\\THESERVER\\EPS\\SERVER\\USERCONFIG\\"
#define MAX_CONT_PERIOD 30
#define MAX_DAY_HIST    15
#define MAX_INTERVAL    (2*60*60)
#define BENCHMARK_SIZE  16

AtpServerSocket::AtpServerSocket(AtpFileServer *rep_server, CCustomerDB *db) : AtpSocket()
{
  m_rep_server = rep_server;
  m_db = db;
}


AtpServerSocket::~AtpServerSocket() {
};

// Return empty string to deny access
CString AtpServerSocket::ServerObtainPassword(CString username) {
  CString pass = "";
  ReceiverInfo info;
  CReceiverInfo *pList;
  pList = m_db->GetReceiverList();
  if (!pList) return pass;
  if(pList->FindReceiver(info, username))
    pass = info.Password;
  m_db->ReleaseReceiverList(pList);
  return pass;
}

void AtpServerSocket::Run() {
  try {
    InitializeServer();
    if (!AuthenticateServer()) return;
    ServerLoginUser();
    ServerLoop();
  } catch (CMyException ex) {
    ex;
    Close();
  }
  ServerLogoutUser();
}
```

```
void AtpServerSocket::RunAnon() {
  try {
    InitializeServer();
    ServerLoop();
  } catch (CMyException ex) {
    ex;
    Close();
  }
}

void AtpServerSocket::UpdateTrafficStats(ReceiverInfo& info) {
  SocketTraffic traffic = GetSocketTraffic();
  info.AvThroughput = (info.AvThroughput*info.LoginCount+
                       traffic.AverageOut*traffic.IntervalOut) /
    (info.LoginCount+traffic.IntervalOut+1);
  info.LoginCount += traffic.IntervalOut;
  if (info.LoginCount > MAX_INTERVAL)
    info.LoginCount = MAX_INTERVAL;
}

void AtpServerSocket::UpdatePingTimes(ReceiverInfo& info) {
  try {
    CTime now = AtpAttrib::ParseTime(AtpAttrib::UnparseTime(CTime::GetCurrentTime()));
    CTime last = AtpAttrib::ParseTime(info.LastConnect);
    CTimeSpan span = now - last;
    if (span.GetTotalMinutes() < MAX_CONT_PERIOD) {
      // assume user has been online in the meantime
      if (now.GetDay() != last.GetDay()) {
        info.TodayLoginTime += 24*60 - (last.GetHour()*60 + last.GetMinute());

        // calc day average
        info.AvDayLoginTime = (info.AvDayLoginTime * info.DayCount +
                               info.TodayLoginTime) / (info.DayCount + 1);
        info.DayCount++;
        if (info.DayCount > MAX_DAY_HIST) info.DayCount = MAX_DAY_HIST;

        info.TodayLoginTime = now.GetHour()*60 + now.GetMinute();
      }
      else
        info.TodayLoginTime += span.GetTotalMinutes();
    }
  } catch (FormatException ex) {
    ex;
    HACKassert("Bad DB entry format (LastConnect)", TRUE);
  }
  info.LastConnect = AtpAttrib::UnparseTime(CTime::GetCurrentTime());
}

void AtpServerSocket::ServerLoginUser() {
  ReceiverInfo info;
  CReceiverInfo * pList;
  pList = m_db->GetReceiverList();
  if (!pList) return;
  if(pList->FindReceiver(info, GetUserName()) == FALSE) {
    m_db->ReleaseReceiverList(pList);
    return;
  }

  m_RegisteredNow = FALSE;
```

```
  if (!info.Registered) {
    m_RegisteredNow = TRUE;
    info.Registered = TRUE;
    info.AvThroughput = 0;
    info.EffThroughput = 0;
    info.AvLoginDuration = 0;
    info.LoginCount = 0;
    info.AvDayLoginTime = 8*60*60;
    info.DayCount = 0;
    info.LastConnect = AtpAttrib::UnparseTime(CTime::GetCurrentTime());
    info.TodayLoginTime = 0;
  }

  info.HostName = m_peerhost;
  UpdatePingTimes(info);

  pList->UpdateReceiver(info);
  m_db->ReleaseReceiverList(pList);
}

void AtpServerSocket::ServerLogoutUser() {
  ReceiverInfo info;
  CReceiverInfo * pList;
  pList = m_db->GetReceiverList();
  if (!pList) return;
  if(pList->FindReceiver(info, GetUserName()) == FALSE) {
    m_db->ReleaseReceiverList(pList);
    return;
  }

  UpdatePingTimes(info);
  UpdateTrafficStats(info);

  pList->UpdateReceiver(info);
  m_db->ReleaseReceiverList(pList);
}

void AtpServerSocket::ServerHandleSend(AtpAttrib& arg) {
  BOOL special = FALSE;
  try {
    special = (!arg[ATP_ARG_SPECIAL].IsEmpty() &&
               AtpAttrib::ParseBool(arg[ATP_ARG_SPECIAL]));
  } catch (FormatException ex) {
    ex;
  }

  CString name = arg[ATP_ARG_NAME];
  if (!special) {
    ServerRejectSend();
    return;
  }

  if (name == ATP_FILE_CONFIG) {
    AtpAttrib config;
    AtpMemFile afl(arg);
    ServerAcceptSend(afl);
    CFile *fl = afl.GetFile();
    fl->SeekToBegin();

    CArchive ar(fl, CArchive::load);
```

```
    config.ParseFile(ar);

    ReceiverInfo info;
    CReceiverInfo * pList;
    pList = m_db->GetReceiverList();
    if (!pList) return;
    if(pList->FindReceiver(info, GetUserName()) == FALSE) {
      m_db->ReleaseReceiverList(pList);
      return;
    }

    pList->GetConfigInfo(info);
    if (info.ComputerConfig.IsEmpty() || !FileExists(info.ComputerConfig)) {
      // create a unique file
      char tmp[MAX_PATH] = "";
      GetTempFileName(CONFIGFILEDIR, "uc", 0, tmp);
      info.ComputerConfig = tmp;
    }
    try {
      CFile *fl = NULL;
      fl = new CFile(info.ComputerConfig, CFile::modeWrite ||
                     CFile::modeCreate);
      CArchive ar(fl, CArchive::store);
      config.UnparseFile(ar);
      ar.Close();
      delete fl;
    } catch (CFileException *e) {
      e->Delete();
    }

    pList->SetConfigInfo(info);
    m_db->ReleaseReceiverList(pList);
    return;
  };

  ServerRejectSend();
  return;
}

void AtpServerSocket::ServerHandleRecv(AtpAttrib& arg) {
  BOOL special = FALSE;
  try {
    special = (!arg[ATP_ARG_SPECIAL].IsEmpty() &&
           AtpAttrib::ParseBool(arg[ATP_ARG_SPECIAL]));
  } catch (FormatException ex) {
    ex;
  }

  CString name = arg[ATP_ARG_NAME];
  if (special) {
    if (name == ATP_FILE_REGISTRATION) {
      AtpMemFile afl(arg);
      ServerAcceptRecv(afl);
      return;
    }

    if (name == ATP_FILE_LIST) {
      AtpMemFile afl(arg);
      CFile *fl = afl.GetFile();
```

```
      CString list = m_rep_server->GetReceiverPackageList(GetUserName());
      fl->Write((LPCSTR) list, list.GetLength());
      fl->Flush();

      ServerAcceptRecv(afl);
      return;
    }

    if (name == ATP_FILE_BENCHMARK) {
      AtpMemFile afl(arg);
      CFile *fl = afl.GetFile();

      char buf[1024];
      for (int i = 0; i < BENCHMARK_SIZE; i++)
        fl->Write(buf, 1024);
      fl->Flush();

      ServerAcceptRecv(afl);
      return;
    }

    ServerRejectRecv();
    return;
  }

  AtpFile *afl = m_rep_server->CreatePackageFile(GetUserName(), arg);
  if (!afl) {
    ServerRejectRecv();
    return;
  }

  try {
    ServerAcceptRecv(*afl);
  } catch (CMyException ex) {
    delete afl;
    ex;
    throw;
  }
  delete afl;
}

///// Debugging utilities /////

#ifdef _DEBUG
void AtpServerSocket::AssertValid() const
{
  AtpSocket::AssertValid();
}

void AtpServerSocket::Dump(CDumpContext& dc) const
{
  AtpSocket::Dump(dc);
}
#endif //_DEBUG


#ifdef  SOURCE_CONTROL_BLOCK
 Project Name: E-PS Client/Server program
 E-PS Inc.
```

```
 Copyright 1996, 1997. All Rights Reserved.
 SUBSYSTEM: ServerProtocol
 $Workfile: atpserver.cpp $
 $Author: Jrward $
 $Revision: 12 $
 $Date: 1/27/97 11:21a $
 $Modtime: 1/27/97 11:21a $

 $History: atpserver.cpp $
 *
 * *****************  Version 12  *****************
 * User: Jrward        Date: 1/27/97     Time: 11:21a
 * Updated in $/atp/ServerProtocol
 * Added VSS history stuff.
#endif    // SOURCE_CONTROL_BLOCK
```

```
// atpsock.h : interface of the CAtpSocket class

#ifndef __ATPSERVER_H__
#define __ATPSERVER_H__

#include "stdafx.h"
#include "atpsock.h"
#include "AtpFileServer.h"
#include "CustomerDB.h"

typedef long AtpConnUID;

class AtpConnection {
public:
  enum AtpState {
    CONNECTING,          // Connection just created
    WAITING,             // Connection established; waiting for transfer
    RECEIVING,           // Transfering data
    PAUSED,          // Transfer paused for some reason
    CLOSING          // Connection will be closed
  };

  AtpConnUID uid;
  AtpState state;
  CString peer_address;
  unsigned int peer_port;
  SocketTraffic *traffic;
  IDFile *idf;           // Valid only if RECEIVING, or PAUSED
};

class AtpServerSocket : public AtpSocket {
  // Inherited public methods:
  // InitializeServer(), AuthenticateServer(), ServerLoop()

public:
  AtpServerSocket(AtpFileServer *rep_server, CCustomerDB *db);
  virtual ~AtpServerSocket();

  void Run();
  void RunAnon();

protected:
  virtual CString ServerObtainPassword(CString username);
  virtual void ServerLoginUser();
  virtual void ServerLogoutUser();

  //virtual void ServerNegotiateBlock(long client_preferred);
  virtual void ServerHandleSend(AtpAttrib& arg);
  virtual void ServerHandleRecv(AtpAttrib& arg);

private:
  void UpdateTrafficStats(ReceiverInfo& info);
  void UpdatePingTimes(ReceiverInfo& info);

protected:
#ifdef _DEBUG
  void AssertValid() const;
  void Dump(CDumpContext& dc) const;
#endif // _DEBUG
```

```
private:
  AtpFileServer *m_rep_server;
  CCustomerDB *m_db;
  BOOL m_RegisteredNow;
};

#endif // __ATPSERVER_H__



#ifdef  SOURCE_CONTROL_BLOCK
 Project Name: E-PS Client/Server program
 E-PS Inc.
 Copyright 1996, 1997. All Rights Reserved.
 SUBSYSTEM: E-parcel.com
 $Workfile: atpserver.h $
 $Author: Jrward $
 $Revision: 6 $
 $Date: 1/27/97 12:13p $
 $Modtime: 1/27/97 12:11p $

 $History: atpserver.h $
 *
 * *****************  Version 6  *****************
 * User: Jrward       Date: 1/27/97    Time: 12:13p
 * Updated in $/atp/Include
 * Added VSS history stuff.
#endif    // SOURCE_CONTROL_BLOCK
```

```
//
// C++ code file
// (c) 1996 ACS
//

#include "stdafx.h"
#include <assert.h>

#include "SSocketControl.h"
#include "atpexcept.h"
#include "sockutil.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

SSocketControl::SSocketControl() {
  m_lsock = NULL;
  m_allow_anon = FALSE;
}

SSocketControl::~SSocketControl() {
  if (m_lsock) delete m_lsock;
  m_lsock = NULL;
}

BOOL SSocketControl::Initialize(AtpFileServer *reps,
                CCustomerDB *db,
                IAtpProgressCallback *pPCB) {
  m_reps = reps;
  m_db = db;
  m_pPCB = pPCB;
  m_lsock = new CListeningSocket(this);
  if (m_lsock->Create(ATP_SERVER_PORT) && m_lsock->Listen())
    return TRUE;
  return FALSE;
}

void SSocketControl::DistributeFiles() {
  AfxBeginThread(StartDistributionThread, this, 0);
  //StartDistributionThread(this);
}

UINT SSocketControl::StartDistributionThread(LPVOID pParam) {
  ((SSocketControl *)pParam)->RunDistributionThread();
  return 0;
}


void SSocketControl::RunDistributionThread() {
  BOOL err;
  ReceiverInfo Receiver, newReceiver;
  CReceiverInfo * pList;
  BOOL bReturn;

  pList = m_db->GetReceiverList();
  if (!pList) return;
```

```
  err = FALSE;
  bReturn = pList->GetFirstReceiver(newReceiver);
  if(FALSE == bReturn)
    err = TRUE;

  while (bReturn) {
    Receiver = newReceiver;
    bReturn = pList->GetNextReceiver(newReceiver);
    if (!err && Receiver.Registered) {
      SOCKET sock = SocketConnectTo(Receiver.HostName, ATP_PORT);
      if (sock != INVALID_SOCKET) {
        ThreadInfo *info = new ThreadInfo();
        info->me = this;
        info->sock = sock;
        AfxBeginThread(StartDistributionSocketThread, info, 0);
      }
    }
    err = FALSE;
    if(FALSE == bReturn)
      err = TRUE;
  }
  m_db->ReleaseReceiverList(pList);
}

UINT SSocketControl::StartDistributionSocketThread(LPVOID pParam) {
  ThreadInfo *info = (ThreadInfo *) pParam;
  info->me->RunServerSocket(info->sock);
  delete info;
  return 0;
}

void SSocketControl::RunDistributionSocket(SOCKET sock) {
  // not used so far -- use RunServerSocketInstead
}


void SSocketControl::ProcessPendingAccept(CAsyncSocket *sock) {
  //for (int i = 0; i < m_todelete.GetSize(); i++)
  //  delete m_todelete[i];
  //m_todelete.RemoveAll();

  CAsyncSocket *atp_sock = new CAsyncSocket();
  if (!sock->Accept(*atp_sock)) {
    delete atp_sock;
    return;
  }

  ThreadInfo *info = new ThreadInfo();
  info->me = this;
  info->sock = atp_sock->Detach();
  delete atp_sock;

  AfxBeginThread(StartSocketThread, info, 0);

  //m_conns.Add(atp_sock->GetConnectionState());
}

UINT SSocketControl::StartSocketThread(LPVOID pParam) {
  ThreadInfo *info = (ThreadInfo *) pParam;
  info->me->RunServerSocket(info->sock);
```

```
  delete info;
  return 0;
}

void SSocketControl::RunServerSocket(SOCKET sock) {
  AtpServerSocket ssock(m_reps, m_db);
  if (!ssock.Attach(sock)) {
    assert(FALSE);
    return;
  }

  if (m_allow_anon)
    ssock.RunAnon();
  else
    ssock.Run();
}

void SSocketControl::ProcessClose(CAsyncSocket *sock) {
}


// JRW: Return state reference for a given connection:
// IN: uid: id of the connection
// OUT: conn: AtpConnection reference

BOOL SSocketControl::GetConnectionState(long uid, AtpConnection*& conn) {
  return FALSE;
}

// JRW: Return pointer to array of all connections.

const CPtrArray* SSocketControl::GetAllConnectionStates() {
  return &m_conns;
}

SocketTraffic SSocketControl::GetSocketTraffic() {
  SocketTraffic traffic;
  return traffic;
}


// End of code

#ifdef  SOURCE_CONTROL_BLOCK
 Project Name: E-PS Client/Server program
 E-PS Inc.
 Copyright 1996, 1997. All Rights Reserved.
 SUBSYSTEM: ServerProtocol
 $Workfile: SSocketControl.cpp $
 $Author: Jrward $
 $Revision: 12 $
 $Date: 1/27/97 11:21a $
 $Modtime: 1/27/97 11:21a $

 $History: SSocketControl.cpp $
 *
 * *****************  Version 12  *****************
 * User: Jrward       Date: 1/27/97    Time: 11:21a
 * Updated in $/atp/ServerProtocol
```

```
 * Added VSS history stuff.
#endif     // SOURCE_CONTROL_BLOCK
```

```
//
// C++ header file
// (c) 1996 ACS
//

#ifndef _SSOCKETCONTROL_H_
#define _SSOCKETCONTROL_H_

#include "stdafx.h"
#include "callback.h"
#include "lstnsock.h"
#include "atpsock.h"
#include "atpfileserver.h"
#include "atpserver.h"
#include "CustomerDB.h"

const ATP_PORT = 608;
const ATP_SERVER_PORT = 609;


// JRW proper "public" on everything ...
class SSocketControl : public ISockCallback,
     public IAtpCallback, public IAtpStateCallback
{
public:
  SSocketControl();
  virtual ~SSocketControl();

public:
  virtual void ProcessPendingAccept(CAsyncSocket *sock);
  virtual void ProcessClose(CAsyncSocket *sock);

  virtual BOOL GetConnectionState(long uid, AtpConnection*& conn);
  virtual const CPtrArray* GetAllConnectionStates();
  virtual SocketTraffic GetSocketTraffic();

  BOOL Initialize(AtpFileServer *reps, CCustomerDB* db, IAtpProgressCallback *pPCB);
  void DistributeFiles();

private:
  struct ThreadInfo {
    SSocketControl* me;
    SOCKET sock;
  };

  static UINT StartSocketThread(LPVOID pParam);
  void RunServerSocket(SOCKET sock);

  static UINT StartDistributionThread(LPVOID pParam);
  void RunDistributionThread();

  static UINT StartDistributionSocketThread(LPVOID pParam);
  void RunDistributionSocket(SOCKET sock);


public:
  BOOL m_allow_anon;

private:
  AtpFileServer *m_reps;
```

```
  CCustomerDB *m_db;
  IAtpProgressCallback *m_pPCB;

  CListeningSocket *m_lsock;
  CPtrArray m_conns;
};


#endif // _SSOCKETCONTROL_H_

// End of headers

#ifdef  SOURCE_CONTROL_BLOCK
 Project Name: E-PS Client/Server program
 E-PS Inc.
 Copyright 1996, 1997. All Rights Reserved.
 SUBSYSTEM: E-parcel.com
 $Workfile: SSocketControl.h $
 $Author: Jrward $
 $Revision: 4 $
 $Date: 1/27/97 12:13p $
 $Modtime: 1/27/97 12:12p $

 $History: SSocketControl.h $
 *
 * *****************  Version 4  *****************.
 * User: Jrward       Date: 1/27/97    Time: 12:13p
 * Updated in $/atp/Include
 * Added VSS history stuff.
#endif    // SOURCE_CONTROL_BLOCK
```

```
// Server1.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "Server1.h"
#include "Server1Dlg.h"

#include "CustomerDB.h"
#include "RepServer.h"
#include "SSocketControl.h"
#include "DistribServer.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#define USERDB_DATASOURCENAME "CustomerDB"

/////////////////////////////////////////////////////////////////////////////
// CServer1App

BEGIN_MESSAGE_MAP(CServer1App, CWinApp)
  //{{AFX_MSG_MAP(CServer1App)
  // NOTE - the ClassWizard will add and remove mapping macros here.
  //    DO NOT EDIT what you see in these blocks of generated code!
  //}}AFX_MSG
  ON_COMMAND(ID_HELP, CWinApp::OnHelp)
  END_MESSAGE_MAP();

/////////////////////////////////////////////////////////////////////////////
// CServer1App construction

CServer1App::CServer1App()
{
  // TODO: add construction code here,
  // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CServer1App object

CServer1App theApp;

/////////////////////////////////////////////////////////////////////////////
// CServer1App initialization

BOOL CServer1App::InitInstance()
{
  if (!AfxSocketInit())
    {
      AfxMessageBox(IDP_SOCKETS_INIT_FAILED);
      return FALSE;
    }

  // Initialize OLE libraries
  if (!AfxOleInit())
    {
      AfxMessageBox(IDP_OLE_INIT_FAILED);
```

```
    return FALSE;
  }

  // Standard initialization
  // If you are not using these features and wish to reduce the size
  //  of your final executable, you should remove from the following
  //  the specific initialization routines you do not need.

#ifdef _AFXDLL
  Enable3dControls();                     // Call this when using MFC in a shared DLL
#else
  Enable3dControlsStatic();      // Call this when linking to MFC statically
#endif

  // Parse the command line to see if launched as OLE server
  if (RunEmbedded() || RunAutomated())
    {
      // Register all OLE server (factories) as running.  This enables the
      //  OLE libraries to create objects from other applications.
      COleTemplateServer::RegisterAll();

      // Application was run with /Embedding or /Automation.  Don't show the
      //  main window in this case.
      return TRUE;
    }

  // When a server application is launched stand-alone, it is a good idea
  //  to update the system registry in case it has been damaged.
  COleObjectFactory::UpdateRegistryAll();


  CCustomerDB *db = new CCustomerDB();
  if (!db->Connect(USERDB_DATASOURCENAME)) {
    AfxMessageBox("Cannot connect to user database.\n");
    return FALSE;
  }

  RepServer *rep = new RepServer();
  //rep.AddDirectory("D:\\idfiles");

  DistribServer *distrib = new DistribServer(db);

  SSocketControl *atp_server = new SSocketControl();
  atp_server->Initialize(distrib, db, NULL);

  CServer1Dlg dlg(rep, atp_server);
  m_pMainWnd = &dlg;
  int nResponse = dlg.DoModal();
  if (nResponse == IDOK)
    {
      // TODO: Place code here to handle when the dialog is
      //  dismissed with OK
    }
  else if (nResponse == IDCANCEL)
    {
      // TODO: Place code here to handle when the dialog is
      //  dismissed with Cancel
    }

  // Since the dialog has been closed, return FALSE so that we exit the
```

```
  //  application, rather than start the application's message pump.
  delete atp_server;
  delete rep;
  delete distrib;
  delete db;
  return FALSE;
}


int WriteErrorLog(char const *,char const *) {
  return ERROR_SUCCESS;
}

#ifdef  SOURCE_CONTROL_BLOCK
 Project Name: E-PS Client/Server program
 E-PS Inc.
 Copyright 1996, 1997. All Rights Reserved.
 SUBSYSTEM: Main Server
 $Workfile: Server1.cpp $
 $Author: Ttonchev $
 $Revision: 7 $
 $Date: 1/30/97 12:55p $
 $Modtime: 1/30/97 12:54p $

 $History: Server1.cpp $
 *
 * *****************  Version 7  *****************
 * User: Ttonchev      Date: 1/30/97    Time: 12:55p
 * Updated in $/atp/ServerApp
 * added a #include for DistribServer (was forgotten; file could not
 * compile)
 *
 * *****************  Version 6  *****************
 * User: Jrward        Date: 1/27/97    Time: 12:05p
 * Updated in $/atp/ServerApp
 * Added VSS History stuff.
#endif    // SOURCE_CONTROL_BLOCK
```

```
// Server1.h : main header file for the SERVER1 application
//

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"       // main symbols

/////////////////////////////////////////////////////////////////////////////
// CServer1App:
// See Server1.cpp for the implementation of this class
//

class CServer1App : public CWinApp
{
public:
    CServer1App();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CServer1App)
    public:
    virtual BOOL InitInstance();
    //}}AFX_VIRTUAL

// Implementation

    //{{AFX_MSG(CServer1App)
        // NOTE - the ClassWizard will add and remove member functions here.
        //    DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////

#ifdef  SOURCE_CONTROL_BLOCK
 Project Name: E-PS Client/Server program
 E-PS Inc.
 Copyright 1996, 1997. All Rights Reserved.
 SUBSYSTEM: Main Server
 $Workfile: Server1.h $
 $Author: Jrward $
 $Revision: 4 $
 $Date: 1/27/97 12:05p $
 $Modtime: 1/27/97 12:04p $

 $History: Server1.h $
 *
 * *****************  Version 4  *****************
 * User: Jrward       Date: 1/27/97    Time: 12:05p
 * Updated in $/atp/ServerApp
 * Added VSS History stuff.
#endif    // SOURCE_CONTROL_BLOCK
```

Having now described a few embodiments of the invention, and some modifications and variations thereto, it should be apparent to those skilled in the art that the foregoing is merely illustrative and not limiting, having been presented by the way of example only. Numerous modifications and other embodiments are within the scope of one of ordinary skill in the art and are contemplated as falling within the scope of the invention as limited only by the appended claims and equivalents thereto.